Peter M.A. Sloot Alfons G. Hoekstra
Thierry Priol Alexander Reinefeld
Marian Bubak (Eds.)

Advances in Grid Computing – EGC 2005

LNCS 3470

European Grid Conference
Amsterdam, The Netherlands, February 2005
Revised Selected Papers

Springer

Lecture Notes in Computer Science 3470

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Peter M.A. Sloot Alfons G. Hoekstra
Thierry Priol Alexander Reinefeld
Marian Bubak (Eds.)

Advances in Grid Computing – EGC 2005

European Grid Conference
Amsterdam, The Netherlands, February 14-16, 2005
Revised Selected Papers

Springer

Volume Editors

Peter M.A. Sloot
Alfons G. Hoekstra
University of Amsterdam, Institute for Informatics, Section Computational Science
Laboratory for Computing, Systems Architecture and Programming
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
E-mail: {sloot, alfons}@science.uva.nl

Thierry Priol
IRISA/INRIA, Campus de Beaulieu
35042 Rennes Cedex, France
E-mail: thierry.priol@irisa.fr

Alexander Reinefeld
Zuse Institute Berlin (ZIB)
Takustr. 7, 14195 Berlin, Germany
E-mail: ar@zib.de

Marian Bubak
AGH University of Science and Technology
Institute of Computer Science and Academic Computer Centre CYFRONET
al. Mickiewicza 30, 30-059 Krakow, Poland
E-mail: bubak@uci.agh.edu.pl

Library of Congress Control Number: 2005928161

CR Subject Classification (1998): C.2.4, D.1.3, D.2.7, D.2.12, D.4, F.2.2, G.2.1

ISSN 0302-9743
ISBN-10 3-540-26918-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-26918-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11508380 06/3142 5 4 3 2 1 0

Preface

> "When the network is as fast as the computer's internal links, the machine disintegrates across the net into a set of special purpose appliances." (George Gilder)

We are proud to present to you the proceedings of the European Grid Conference 2005, held at the Science Park Amsterdam during February 14–16.

The aim of the European Grid Conference was to be the premier event on Grid computing in Europe in 2005, focusing on all aspects of Grid computing and bringing together participants from research and industry. EGC 2005 was a follow-up of the Across Grids Conferences held in Santiago de Compostela, Spain (2003) and in Nicosia, Cyprus (2004).

We decided to have three main tracks during this conference: one with peer-reviewed scientific contributions, one with presentations from business and industry, and one event track with presentations from European and national Grid projects.

In order to guarantee high-quality proceedings, we put extensive effort into reviewing the scientific papers and processing the proceedings. We received over 180 papers from which, after peer review by 2–3 reviewers each, we selected 70 for oral presentations and 52 for poster presentations during the scientific tracks. In this book you find the final versions of these accepted papers.

After the conference opening by the Dean of the Faculty of Science of the University of Amsterdam, Prof. Dr. K.J.F Gaemers, we enjoyed a series of inspiring keynote lectures and two parallel scientific tracks over three days.

The keynote addresses were given by:

- Domenico Laforenza "Towards a Next Generation Grid: Learning from the Past, Looking into the Future"
- Bob Hertzberger "e-Science and Grid"
- Wolfgang Boch "Moving Grids from Science into Industry and Business – Challenges of EU Grid Research"
- Peter Coveney "Real Science on Computational Grids"
- Thierry Priol "Objects, Components, Services for Grid Middleware: Pros and Cons"
- Malcom Atkinson "Lessons Learned Building OGSA-DAI — Middleware for Distributed Data Access"
- Carol Goble "Semantic(Grid services)+(Semantic Grid)Services"
- Carl Kesselman "Managing Work Across Virtual Organizations: The GriPhyN Virtual Data System"

We would like to express our sincere thanks to the invited speakers who delivered such high-quality lectures at EGC 2005.

The scientific programme of the conference was organized along the following tracks:

- Applications
- Architecture and Infrastructure
- Resource Brokers and Management
- Grid Services and Monitoring
- Performance
- Security
- Workflow
- Data and Information Management
- Scheduling Fault-Tolerance and Mapping

This conference would not have been possible without the support of many people and organizations that helped in various ways to make it a success.

First of all we would like to thank the authors who took the effort to submit so many high-quality papers. We thank the Programme Committee for their excellent job in reviewing the submissions and thus guaranteeing the quality of the conference and the proceedings. We thank Lodewijk Bos and his staff for their practical assistance and support. Many thanks go to Coco van der Hoeven for her secretarial work. Dick van Albada, Berry Vermolen, Dennis Kaarsemaker and Derek Groen are acknowledged for their punctuality in preparing the proceedings.

We thank our sponsors for their financial support: the Board of the University of Amsterdam, the Science Faculty and the Institute for Informatics. Finally we thank the Dutch Science Foundation NWO, Section Exact Sciences.

February 2005

P.M.A. Sloot, A.G. Hoekstra, T. Priol, A. Reinefeld and M. Bubak

Organization

Overall Event Chair

- Prof. Dr. P.M.A. Sloot, University of Amsterdam, The Netherlands

Scientific Committee

- Dr. A.G. Hoekstra (chair), University of Amsterdam, The Netherlands
- Dr. M. Bubak, AGH, Cracow, Poland
- Dr. Th. Priol, IRISA, Paris, France

Industrial and Business Board

- Drs. A. Emmen (chair), Genias Benelux, The Netherlands
- Dr. A. Osseyran, Sara Computing and Networking Services, Amsterdam, The Netherlands
- Dr. W. Boch, European Commission, Brussels
- Dr. A. Reuver, IBM, The Netherlands

Special Events Board

- Drs. L. Bos (chair), MC-Consultancy, The Netherlands
- Prof. Dr. L.O. Hertzberger, University of Amsterdam, The Netherlands
- Prof. Dr. M. Turala, Institute of Nuclear Physics, Cracow, Poland
- Dr. K. Baxevanidis, European Commission, Brussels

Local Steering Committee

- Prof. Dr. W. Hoogland, Dean of the Faculty of Science, University of Amsterdam
- Prof. Dr. B. Noordam, Director of the FOM Institute for Atomic and Molecular Physics AMOLF, Amsterdam, The Netherlands
- Prof. Dr. J.K. Lenstra, Director of the Center for Mathematics and Computer Science, Amsterdam, The Netherlands
- Prof. Dr. K. Gaemers, Director of the National Institute for Nuclear Physics and High Energy Physics, Amsterdam, The Netherlands
- Prof. Dr. E. P.J. van de Heuvel, Director of the Astronomical Institute "Anton Pannekoek", University of Amsterdam, The Netherlands

Programme Committee

- Albada, G.D. van — University of Amsterdam, The Netherlands
- Abramson, D. — Monash University, Australia
- Andrzejak, A. — ZIB Berlin, Germany

- Badia, R. — Technical University of Catalonia, Spain
- Baker, M. — University of Portsmouth, UK
- Bal, H. — Free University Amsterdam, The Netherlands
- Baraglia, R. — ISTI-CNR, Italy
- Beco, S. — DATAMAT S.p.A., Italy
- Benkner, S. — University of Vienna, Austria
- Bilas, A. — ICS-FORTH, Greece
- Breton, V. — Laboratoire de Physique Corpusculaire de Clermont-Ferrand, France
- Brezany, P. — University of Vienna, Austria
- Bubak, M. — Inst. of Comp. Sci., and Cyfronet, Poland
- Buyya, R. — University of Melbourne, Australia
- Chun-Hsi Huang — University of Connecticut, USA
- Corbalan, J. — Technical University of Catalonia, Spain
- Cunha, J. — New University of Lisbon, Portugal
- Danelutto, M. — University of Pisa, Italy
- Deelman, E. — ISI, Univ. of Southern California, USA
- Dikaiakos, M. — Univ. of Cyprus, Cyprus
- DiMartino, B. — Second University of Naples, Italy
- Epema, D. — Delft University of Technology, The Netherlands
- Erwin, D. — Forschungszentrum Jülich GmbH, Germany
- Fisher, S. — RAL, UK
- Foster, I. — Argonne National Laboratory
- Fox, G. — Univ. of Indiana, USA
- Fusco, L. — ESA, Italy
- Gomez, A. — CESGA, Spain
- Gorlatch, S. — University of Muenster, Germany
- Guisset, P. — CETIC, Belgium
- Hluchy, L. — Slovak Academy of Science, Slovakia
- Hoekstra, A. — Univ. of Amsterdam, The Netherlands
- Houstis, E. — University of Thessaly, Greece
- Jones, R. — CERN, Switzerland
- Kesselman, C. — USC/Information Sciences Institute, USA
- Kielmann, Th. — Free University Amsterdam, The Netherlands
- Kornmayer, H. — KZK, Germany
- Kranzlmüller, D. — Johannes Kepler University Linz, Austria
- Kunszt, P. — CERN, Switzerland
- Laat, C. de — University of Amsterdam, The Netherlands
- Laforenza, D. — ISTI-CNR, Italy
- Marco, J. — CSIC, Santander, Spain
- Markatos, E. — ICS-FORTH, Greece
- Marten, H. — Forschungszentrum Karlsruhe GmbH, Germany
- Matyska, L. — Masary University, Czech Republic
- Meyer, N. — Poznan Supercomputing Center, Poland
- Moreau, L. — Univ. of Southampton, UK
- Morin, C. — IRISA/INRIA, France

- Nemeth, Z. — MTA SZTAKI Computer and Automation Research Institute, Hungary
- Novotny, J. — MPI für Gravitationsphysik, Germany
- Orlando, S. — University of Venice, Italy
- Pazat, J.-L. — IRISA, France
- Perez, C. — INRIA, France
- Perrott, R. — Queen's University Belfast, UK
- Pflug, G. — University of Vienna, Austria
- Priol, T. — INRIA/IRISA, France
- Rana, O. — Cardiff University, UK
- Reinefeld, A. — ZIB Berlin, Germany
- Rodero, I. — Technical University of Catalonia, Spain
- Romberg, M. — Forschungszentrum Jülich GmbH, Germany
- Sakellariou, R. — Univ. of Manchester, UK
- Senar, M. — Univ. Autònoma de Barcelona, Spain
- Sloot, P. — Univ. of Amsterdam, The Netherlands
- Szymanski, B. — Rensselaer Polytechnic Institute, USA
- Talia, D. — Università della Calabria
- Trancoso, P. — Univ. of Cyprus, Cyprus
- Turner, S.J. — Nanyang Technological University, Singapore
- Wismüller, R. — TU München, Germany
- Ziegler, W. — Fraunhofer Institute for Algorithms and Scientific Computing, Germany

Sponsoring Organizations

- University of Amsterdam, The Netherlands
- Dutch Science Foundation NWO, Section Exact Sciences, The Netherlands
- SciencePark Amsterdam, The Netherlands

Local Organizing Committee

- Coco van der Hoeven (University of Amsterdam, The Netherlands)
- Dick van Albada (University of Amsterdam, The Netherlands)
- Berry Vermolen (University of Amsterdam, The Netherlands)
- Derek Groen (University of Amsterdam, The Netherlands)
- Dennis Kaarsemaker (University of Amsterdam, The Netherlands)
- Lodewijk Bos (MC-Consultancy, The Netherlands)

Table of Contents

Author Index

Telemedical Applications and Grid Technology

Georgi Graschew[1], Theo A. Roelofs[1], Stefan Rakowsky[1], Peter M. Schlag[1], Sahin Albayrak[2], and Silvan Kaiser[2]

[1] Surgical Research Unit OP 2000,
Max-Delbrueck-Centrum and Robert-Roessle-Klinik,
Charité – University Medicine Berlin,
Lindenberger Weg 80, D-13125 Berlin, Germany
[2] DAI-Labor, Agent Technologies in Business Applications and Telecommunication,
Technical University Berlin, Salzufer 12,
D-10587 Berlin, Germany

Abstract. In recent years different institutions have launched several telemedicine projects which aimed to encourage the Euro-Mediterranean co-operation. The creation of a Virtual Euro-Mediterranean Hospital aiming to facilitate the interconnection of the various services through real integration has been recommended. Therefore Grid becomes inevitable for successful deployment of these services. Existing Grid Engines provide basic computing power needed by today's medical analysis tasks but lack other capabilities needed for communication and knowledge sharing services envisioned. When it comes to heterogeneous systems to be shared by different institutions especially the high level system management areas are still unsupported. Therefore a Metagrid Engine is needed that provides a superset of functionalities across different Grid Engines and manages strong privacy and Quality of Service constraints at this comprehensive level.

1 Introduction

1.1 The EMISPHER Network for Telemedical Applications

The EMISPHER project (Euro-Mediterranean Internet-Satellite Platform for Health, medical Education and Research, see www.emispher.org/) is dedicated to telemedicine, e-Health and medical e-learning [1]. During its implementation over the last two years, EMISPHER has deployed and put in operation a dedicated internet-satellite platform consisting of currently 10 sites in 5 MEDA countries (Casablanca, Algiers, Tunis, Cairo and Istanbul) and 5 EU countries (Palermo, Athens, Nicosia, Clermont-Ferrand and Berlin).

1.2 From EMISPHER Towards the Deployment of a Virtual Euro-Mediterranean Hospital

The EMISPHER network serves as a basis for the development and deployment of a Virtual Hospital for the Euro-Mediterranean region. The Virtual Euro-Mediterranean

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1 – 5, 2005.

Hospital (VEMH) aims to facilitate and accelerate the interconnection and interoperability of the various services, being developed by different organizations at different sites, through real integration.

1.3 Services of the VEMH

- Euro-Mediterranean Medical University
  improved qualification by exchange and standardization of educational modules
- Real-time Telemedicine
  improved quality of patient care and qualification of staff
- Medical assistance
  improved continuity of care to stimulate tourism
- Implementation of Evidence-based medicine
  improved disease management and individual therapies
- Fellowship programmes for young professionals
  improved qualifications in multi-cultural and inter-disciplinary settings

1.4 Methodologies for the VEMH

- Medical-needs-driven instead of technology-driven!!!
- New management tools for virtual medical communities
  - *trust- and synergy-building measures*
  - *balance between technological and interpersonal skills*
- Management of clinical outcomes
  improved implementation of evidence-based medicine
- Modular architecture
- Integration of different telemedical solutions in one platform to support many different medical services (matrix structure)
  - *individual user needs (as matrix columns)*
  - *technical solutions (as matrix rows)*
- Data security & Patient's privacy

1.5 Communication Infrastructures for a Virtual Hospital

The communication infrastructure of such a Virtual Euro-Mediterranean Hospital should integrate satellite-based networks like EMISPHER with suitable terrestrial channels. Due to the distributed character of the Virtual Euro-Mediterranean Hospital, data, computing resources as well as the need for these are distributed over many sites in the Virtual Hospital. Therefore Grid becomes inevitable for successful deployment of services like acquisition and processing of medical images, data storage, archiving and retrieval, data mining (especially for evidence-based medicine). A more detailed description of the envisioned telemedical services can be found in [2].

Giving access to distributed services in a wide network of connected institutions the system shall integrate domain knowledge, powerful computing resources for analytical tasks and means of communication with partners and consultants in a trusted and secure user tailored support system.

2 Approach

2.1 Metagrid Services

Here we introduce our view of an agent-based Metagrid Service Engine (MGSE), which implements an additional software layer between proprietary Grid engines and the applications and therefore integrates the different approaches. It will make a global Grid across OEM-boundaries become reality.

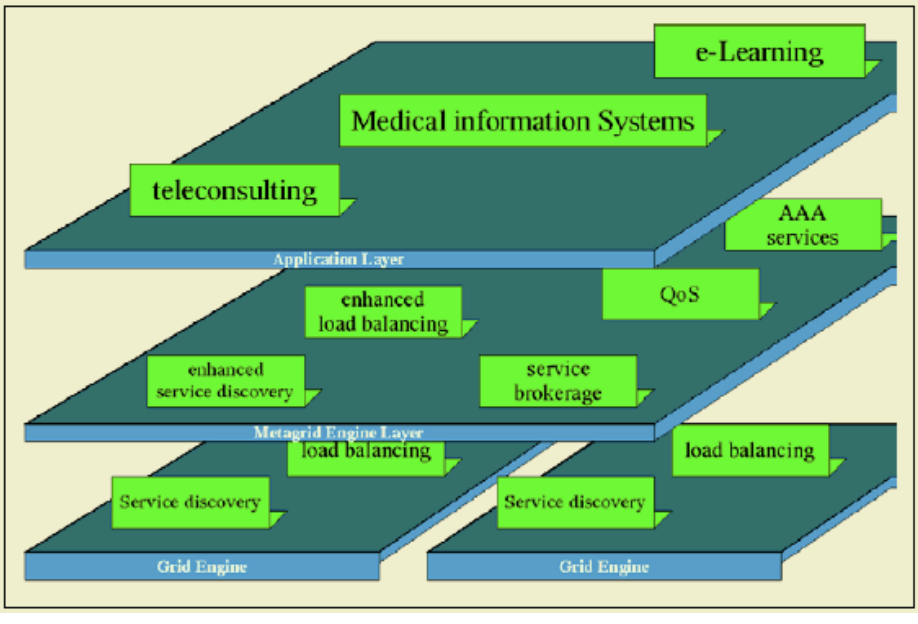


Fig. 1. Agent-based Metagrid Service Engine (MGSE)

The Metagrid Services should address the main issues of today's Grid Computing software. Low level Grids like the SUN Grid Engine provide scalable high performance Grids but have several requirements and shortages that need to be taken care of. First they need homogeneous Grid nodes because scheduled tasks contain only scripts designed to call the programs needed for the tasks.

Furthermore the low level design does not handle AAA aspects too well. When applying the ideas of inter-organizational Grids this becomes a very important issue, as perceived by the Global Alliance. Focusing heavily in this partial area, the Open Grid Service Architecture (OGSA) uses Web Services and sophisticated Authentication and Authorization techniques. Web Services allow a platform independent approach, combined with the included security mechanisms this becomes an important basis for the Global Grid vision [3].

2.2 Mobile Code

We envision the use of mobile code in future Grids which allows service creation and deployment on arbitrary nodes of a Grid giving a flexibility unknown by today's Grid technology. Services can be created and distributed in a Grid through mobile code, severely reducing the need for software installation in the Grids nodes.

The main objective is on the one hand an integration of low level concepts like the SUN Grid Engine in wide scale management like the OGSA and on the other hand bringing more flexibility to systems based on the OGSA framework by integrating features like dynamic service distribution, mobile code, etc. The ultimate goal being the ability to distribute tasks over a secure and organization spanning dynamic Grid.

2.3 Dynamic Grid

By using platform independent software Grids can be extended to large scale networks, allowing the flexible use of computing resources from a vast number of Grid nodes. Here support for dynamic Grid structures becomes an important point. Meta-Grid Services have to be able to tolerate changes in the managed Grid, like the addition or removal of nodes at runtime. Fallback mechanisms have to go to work when tasks cannot be fulfilled because nodes they where assigned to drop out without returning the appropriate task results.

Sub Grids can be put to best use by assigning tasks through intelligent Meta-Grid Services, these can differentiate the individual needs of a task and assign it to an applicable node or sub Grid, e.g. by assigning mathematical computations to high power SUN Grids while directing low priority tasks to other nodes (which possibly demand lower prices).

Integrating meta level accounting is an important Meta-Grid Service, empowering a Meta-Grid to ensure full accounting of scheduled tasks, including the use of sub Grids and other integrated legacy systems. These become immensely important when thinking about the inter-organizational aspects of global Grid structures.

2.4 Experimental Environment

In co-operation between Sun Microsystems and the Technical University Berlin a Grid testbed consisting of seven multiprocessor nodes based on heterogeneous hardware configurations has been set up. All nodes are interconnected by a Gigabit testbed. The systems are configured to run Linux or Solaris operating systems.

3 Conclusions and Perspectives

For successful deployment of the various medical services in the Virtual Euro-Mediterranean Hospital, the development and implementation of Health Grid technology appears crucial.

Subsequently, the implementation of this new technology might trigger a critical evaluation and adaptation / optimization of the medical workflow and corresponding decision-making trees.

References

[1] Graschew, G., Roelofs, T.A., Rakowsky, S., Schlag, P.M., Überbrückung der digitalen Teilung in der Euro-Mediterranen Gesundheitsversorgung – das EMISPHER-Projekt, In: A. Jäckel (Hrsg.) Telemedizinführer Deutschland, Ober-Mörlen, Ausgabe 2005, p. 231-236.

[2] http://whitepaper.healthgrid.org/

[3] Foster, I., Kesselman, C., Nick, J., Tuecke, S., An Open Grid Services Architecture for Distributed Systems Integration, 2002, http://www.globus.org/research/papers/ogsa.pdf.

Statistical Modeling and Segmentation in Cardiac MRI Using a Grid Computing Approach

Sebastian Ordas[1], Hans C. van Assen[2], Loic Boisrobert[1], Marco Laucelli[3], Jesús Puente[3], Boudewijn P.F. Lelieveldt[2], and Alejandro F. Frangi[1]

[1] Computational Imaging Laboratory, Universitat Pompeu Fabra, Barcelona, Spain
sebastian.ordas@upf.edu
[2] Division of Image Processing, Department of Radiology,
Leiden University Medical Center, Leiden, The Netherlands
[3] GridSystems S.A., Palma de Mallorca, Spain

Abstract. Grid technology is widely emerging as a solution for wide-spread applicability of computerized analysis and processing procedures in biomedical sciences. In this paper we show how a cardiac image analysis task can substantially benefit from Grids, making use of a middleware service tailored to the needs of common application tasks. In a first part we describe a methodology for the construction of three-dimensional (3D) statistical shape models of the heart, from a large image database of dynamic MRI studies. Non-rigid registration is needed for the automatic establishing of landmark correspondences across populations of healthy and diseased hearts; but when dealing with large databases, the computational load of current algorithms becomes a serious burden. Our Grid service API provided an easy way of taking benefit from our computing resources, by allowing for pipelining the distributed and non-distributed steps of the algorithm. As a second part of this work we show how the constructed shape models can be used for segmenting the left ventricle in MRI studies. To this aim we have performed an exhaustive tuning of the parameters of a 3D model-based segmentation scheme, also in a distributed way. We run a series of segmentation tests in a Monte Carlo fashion, but only making use of the Grid service web portal, as this time the pipeline was simpler. Qualitative and quantitative validation of the fitting results indicates that the segmentation performance was greatly improved with the tuning, combining robustness with clinically acceptable accuracy.

1 Introduction

Cardiac imaging acquisition technology is being developed vertiginously by means of tremendous technical progresses. These advances include an increased spatial and temporal resolution, motion, exposure and scanning time reduction, and the availability of functional imaging sequences for perfusion, viability and flow assessment. Nevertheless, the increased amount of data originated from state-of-the-art imaging examinations could not been favored in a similar way from robust computerized quantitative analysis tools appropriate for the routine clinical practice. Owing to this necessity, cardiac image analysis has become quite an active field of research. An inevitable step before pursuing any kind of quantitative and/or functional analysis is the automated segmentation

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 6–15, 2005.

of the cardiac chambers. In the last few years, many model-based approaches for image segmentation have contributed to the quite evolving field of medical image analysis. The rationale behind these methods is to analyze the image in a top-down fashion. A generic template model of the structure of interest is deformed to accommodate for the clues provided by image information. For a cardiac application, it is possible to learn the shape statistics of a heart from a population of healthy and/or diseased hearts, and construct a compact and specific anatomical model. Statistical model-based approaches work in this way, and are able to provide constraints that allow for efficiently handling situations with substantial sparse or missing information. Statistical models of shape (ASM) [1] and appearance (AAM) [2] variability are two model-driven segmentation schemes very popular in medical image analysis. In building these frameworks, a set of segmentations of the shape of interest is required, as well as a set of corresponding landmarks defined over them. An ASM comprises a shape and an appearance model. The former primarily holds information about the shape and its allowed variations in a Point Distribution Model (PDM), determined by a Principal Component Analysis (PCA). The latter is responsible of learning grey-level patterns from the training set image data, that are to be compared against those identified in a new (unknown) image during the fitting stage. The algorithm therefore consists of an iterative process in which the appearance model looks into the image for new candidate positions to deform the shape model, and the shape model applies statistical geometric constraints in order to keep the deformation process always within legal statistical limits. First approaches like [3, 4] and [5] have evidenced an encouraging performance in segmenting the left ventricle in 3D data sets of MR/US and MR/CT, respectively.

In the work that we present here, we describe the methodology employed to automatically construct 3D-PDMs from large databases of cardiac dynamic studies [6], as well as the Grid computing approach that has enabled their construction [7]. The ultimate goal of such a representation is to model the geometry of the heart ensuring that the final model will gather representative statistics of the studied population. The statistical geometric constraints implied by such a representation can provide *a priori* knowledge for different model-based segmentation approaches. To this aim, we used the constructed 3D-PDMs within the 3D-ASM segmentation algorithm of van Assen *et al.* [8], that uses a fuzzy inference system in the appearance model. By running the segmentation tests with three different shape models, we aimed to explore to what extent the use of our method for automatically building shape models does really improve the segmentation performance. The way our Grid middleware is designed to work allowed for quite an easy and general methodology for setting-up, running, and post-processing the results. In the following sections we provide the general ideas behind the shape model construction methodology and the parametric tuning of the segmentation algorithm, making emphasis on our experience with the use of Grid computing.

2 Construction of the Statistical Shape Models

2.1 Training Data Set

We built the statistical shape models from a data set comprising 90 MRI studies of healthy and diseased hearts. They were acquired from CETIR Sant Jordi Cardiovascu-

lar MR Centre (Barcelona, Spain) using a General Electric Signa CV/i, 1.5 T scanner (General Electric, Milwaukee, USA) with the FIESTA protocol. The slice thickness was 8–10 mm with an in-plane pixel resolution of $1.56 \times 1.56\ mm^2$. Studies correspond to 21 healthy subjects and 74 patients suffering from common cardiac pathologies, randomly selected from those acquired during year 2002 in the clinical center. Therefore, the types and relative proportion of pathologies were representative of typical examinations. Manual delineations corresponding to five phases were considered. We were thus able to construct both single-phase (90 samples) and multi-phase (450 samples) shape models.

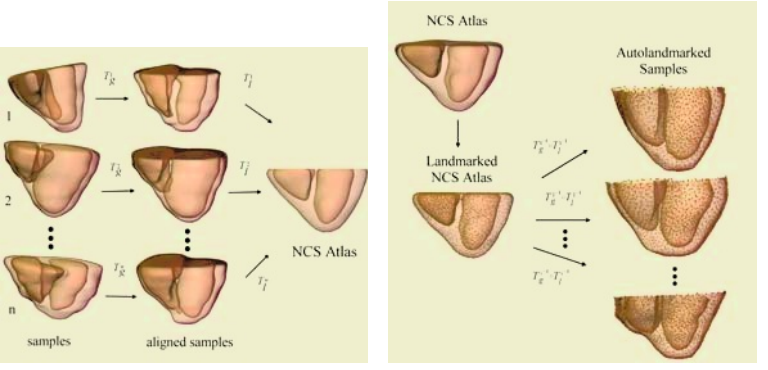
2.2 Grid Service and Cluster Facility

Our Grid middleware platform is the InnerGrid Nitya developed by GridSystems (Mallorca, Spain), running on a 45-node dual Xeon (2.8 GHz CPU, 2 GHz RAM) cluster, under Linux RedHat 9, accessible through a web interface (desktop portal) and from an API. As a whole, the cluster represents more than 103 GFlops of computing power. As each node in the cluster provides 2 Agents to the Grid, a total of 90 Agents were available for tasks distribution.

2.3 Establishing Point Correspondences

The general layout of the method is to align all the images of the training set to an atlas that can be interpreted as a mean shape. Once all the necessary transformations are obtained, they are inverted and used to propagate any number of arbitrarily sampled landmarks on the atlas, to the coordinate system of each subject. In this way, while it is still necessary to manually draw the contours in each training image, our technique reliefs from manual landmark definition and for establishing the point correspondence across the training set. The method can easily be set to build either 1- or 2-chamber models. Moreover, its generality allows for using it with other modalities (e.g. SPECT, CT) and organs with shape variability close to that of the heart (e.g. liver, kidneys). A detailed description of the method can be found in [6], but can be summarized as follows:

1. The manually drawn contours in the training set are converted into labelled shapes simply by flood-filling each (closed) sub-part with a different scalar value.
2. The labelled shapes are aligned through a global transformation (rigid registration with nine degrees of freedom: translation, rotation, and anisotropic scaling) to a Reference Sample (RS) randomly chosen from the training set. The RS is therefore considered as the first atlas estimate.
3. A new atlas is constructed by shape-based averaging of the aligned shapes. This is performed by averaging the images in their distance transform domain, and defining a new labelled shape by considering the zero iso-surface of each sub-part separately.
4. To minimize the bias introduced by the choice of the RS, steps 2 and 3 are repeated until the atlas becomes stable. At this point, the atlas is said to be in a Reference Coordinate System (RCS).
5. Subsequently, each rigidly aligned shape is locally deformed (using non-rigid registration) in order to accommodate to the RCS atlas.



(a) (b)

Fig. 1. (a) Final transformations. A set of final global (T_g) and local (T_l) transformations can take any sample shape of the training set, to the NCS atlas coordinate system. (b) Landmark propagation. Once the final global and local transformations are obtained, they are inverted and used to propagate any number of arbitrarily sampled landmarks on the NCS atlas, to the coordinate system of the original samples

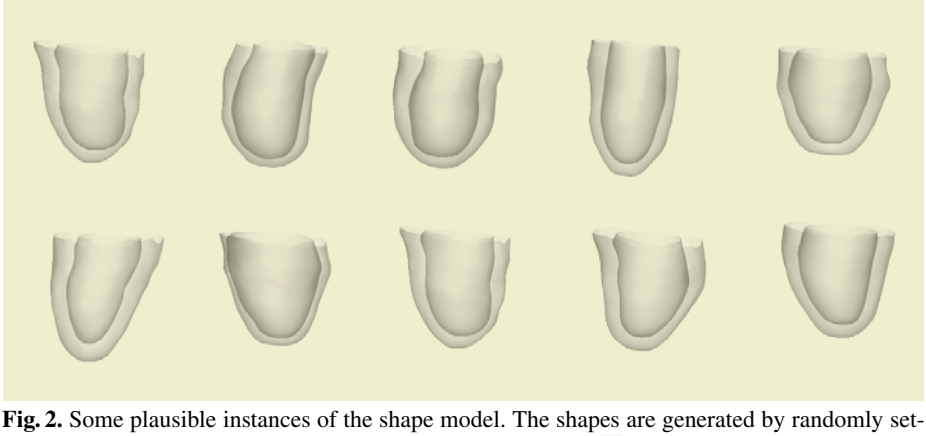


Fig. 2. Some plausible instances of the shape model. The shapes are generated by randomly setting the shape parameters within the limits of ± 3 SD (SD=$\sqrt{\lambda_i}$) from their mean values in the training set

6. The obtained local transformations are averaged and the resulting transformation is applied to the RCS atlas. The new atlas is said to be in a Natural Coordinate System (NCS) and is unique regardless the RS election [6].
7. A new set of global and local transformations are recalculated in the same way as in steps 2 and 5 (Fig. 1(a)).
8. Finally, any automatically generated landmarks in the NCS atlas can be propagated to the training shapes through the transformations in 7 (Fig. 1(b)).

9. In order to build the statistical shape models, the autolandmarked shapes are normalized with respect to a reference coordinate frame, eliminating differences across objects due to rotation, translation and size. Once the shape samples are aligned, the remaining differences are solely shape related, and PCA can be performed. In Fig. 2 some examples of valid shapes generated by the 1-chamber model are illustrated.

2.4 Grid Computing Approach

For this implementation it was necessary to use the Grid middleware API, by means of a *perl* script. Some parts of the PDM construction methodology were distributed and others not. The preprocessing (image labelling, shape-based interpolation) and postprocessing (shape-based averaging, landmark propagation, PCA) tasks, were performed by a single Agent. All the registrations (rigid and non-rigid) were distributed among the Agents. With this facility, for instance, the construction of an atlas of 450 samples with 5 iterations takes approximately 50 minutes. Considering that the average processing time for a rigid registration of one sample to an atlas is 4.5 minutes and that the pre- and post- processes tasks take about 16 minutes, this time would represent one week of calculation for a single equivalent CPU. For the non-rigid registration of the same 450 samples, the distributed process lasted 18 hours, while for a single equivalent CPU it would have taken around 1480 hours (two months).

3 Segmentation Algorithm

3.1 Appearance Model

In order to deform the shape model, candidate points are collected from the image data using a decision scheme based on the Takagi-Sugeno Fuzzy Inference System (FIS) [9]. For a complete description of this decision scheme, we refer to [8]. The Fuzzy C-Means (FCM) within the FIS yields three membership distributions for the tissues: air, myocardium and blood. In the inference part of the FIS, crisp values per pixel are derived based on the fuzzy membership distributions. This is performed using two kinds of thresholds:

1. **Gray level threshold**: first, a threshold is placed in the membership distributions, marking a part that is attributed to a tissue class irrespective of its membership value for that class. All pixels with a gray value below the threshold are assigned to the air-class. The position of the threshold is determined as a proportion between tissue class centers. The threshold is placed at a preset proportion between the tissue class centers of the air and the myocardium tissue classes, resulting from the FCM.
2. **Membership degree thresholds**: the gray level threshold above divides the membership distributions from the FCM in two parts, assigning tissue labels to the lower part. The remaining part is classified using membership threshold levels. Thus, pixels with a gray value in this part are assigned the tissue label of the class with the highest membership value at that particular gray value, provided this membership value is above the threshold for the tissue. Pixels with gray values whose highest membership value does not reach the corresponding tissue membership threshold, are left unclassified.

3.2 Matching Procedure

In summary, the 3D-ASM matching can be described as follows. The mean shape model is placed in the target data set with a given initial scale and pose. The image planes intersect the model's sub-part surface meshes, yielding stacks of 2D contours. These contours are composed of the intersections of the image planes with individual mesh triangles. Candidate displacements (update-vectors) are propagated to the nodes in the mesh. To facilitate through-plane motion in the model, they are projected on the model surface normals, which also have a component perpendicular to the imaging planes. The current model state is aligned to the proposed model state resulting from the model update information using the method of Besl and McKay [10] for 3D point clouds, effectively eliminating scaling, translation and rotation differences. The residual shape differences are projected on the shape model subspace, yielding a model parameter vector. The steps above are repeated either for a fixed number of iterations, or until convergence is achieved.

3.3 Parametric Optimization

The fitting algorithm can easily be set to work either with the 1- or 2-chamber representations built in the previous sections. Nevertheless, we have evidenced that the segmentation results with the latter were not good enough in the area around the RV apex. The confounding point candidates provided by this complex region of the data set hampered the overall performance of the algorithm, by not allowing the model to stretch towards that position. We realized that this effect was shadowing the differences in segmentation performance that we were investigating with the use of different shape model construction methodologies. Therefore, the results that we report in the following sections only correspond to the 1-chamber shape model.

Parameters Related to the Shape Model. For every intersection of the model mesh with an image plane, a model update is computed as explained before. For every single update, the possibility exists that a non-optimal or even an erroneous position results. To diminish the effects of erroneous updates, the update itself, which acts as a force on a model surface, is smeared out over a small region on the model surface around the source location. Thus, faulty or less reliable updates can be compensated for by a number of neighboring updates. The contribution of a single model update to other updates is weighted with a Gaussian kernel, with the actual weight depending on the geodesic distance along the mesh edges to the source location of the update. To limit the extent of the propagation of the updates, propagation is stopped either after a fixed number of edges, or when the weight attributed to a particular update is below a preset threshold. The actual values for the standard deviation of the kernel (*sigma*, σ), the propagation level limit (*extent*, χ), and the number of standard deviations (*beta*, β) that each shape model parameter is allowed to vary, are the three shape-related parameters to optimize (see Table 1).

Parameters Related to the Appearance Model. The three membership thresholds mentioned in Sec. 3.1 (for air, myocardium, and blood) constitute the three parameters

Table 1. Appearance model parameters, ranges and optimal values

Tissue	Lower	Upper	Step size	ED optimal	ES optimal
Appearance model parameters					
Blood	0.1	0.5	0.1	0.2	0.4
Myocardium	0.05	0.30	0.05	0.05	0.05
Air	0.3	0.7	0.1	0.5	0.5
Shape model parameters					
Sigma, σ	3	9	1	6	4
Extent, χ	1	5	1	2	4
Beta, β	1	3	1	2	1

to update (*t1*, *t2*, *t3*) in relationship with the appearance model. Their tuning ranges are specified in Table 1.

Fixed Settings. The 3D-ASM was set to run for a fixed number of iterations (N=100) using 60 modes of variation (more than 95% of the total variance of the three shape models tested). Always, the shape resulting from the final iteration was taken for assessment of its point-to-surface (P2S) error with respect to its manual segmented counterpart. The optimal settings for the algorithm where chosen based on the unsigned P2S distance measures averaged over the complete evaluation database.

4 Fitting Experiments

4.1 Evaluation Data Set

The data set used for the segmentation tests comprised 30 subjects. Fifteen were short-axis scans of healthy volunteers acquired at the Leiden University Medical Center (Leiden, Netherlands) using the balanced FFE-protocol on a Philips Gyroscan NT Intera, 1.5 T MR scanner (Philips Medical Systems, Best, Netherlands). The slice thickness was 8 mm, with a slice gap of 2 mm and in-plane pixel resolution of 1.36×1.36 mm^2. The other fifteen studies corresponded to the same clinical center mentioned in Sec. 2.1. Selected patients suffered from common cardiac diseases: myocardium infarction (10), hypertrophy (2), and pericarditis (3). Expert segmentations were manually drawn along the epicardial (*LV-epi*) and endocardial (*LV-endo*) borders of the left ventricle (LV), at two phases of the cardiac cycle: End Systole (ES) and End Diastole (ED). Manual segmentations usually have an average intra- and inter-observer variability in the range of 1–2 mm. Nevertheless, they generally constitute the gold standard for assessing the achieved accuracy.

4.2 Grid Computing Approach

For setting up the experiments, 60 data sets (30 patients at ED and ES) and 6 shape models (three construction methods and a different shape model for ED and ES) were uploaded to the server. The tuning of the parameters that affect the appearance model and the *beta* parameter, was run first. The remaining parameters were fixed at $\chi = 3$ and $\sigma = 6$ (values beforehand expected to be not far from optimal). The process produced approximately 64,800 results and lasted 1.5 days. The optimal set of parameters of this

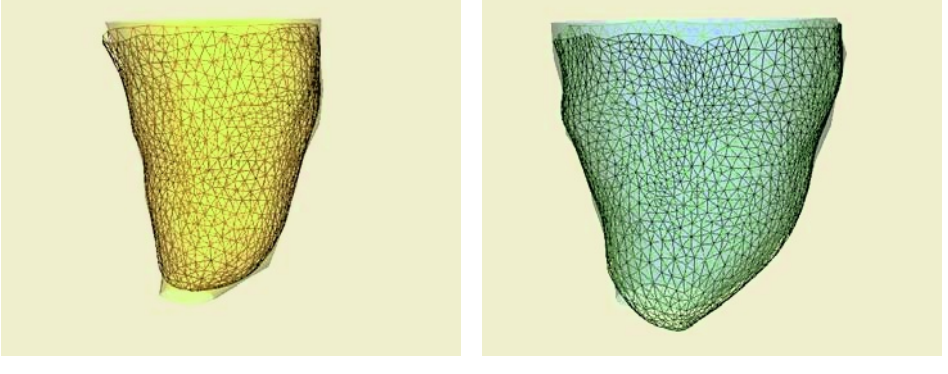
first step were used in a second run for tuning the other two parameters that affect the shape model. This second run lasted 0.9 days and produced 5,670 result files. In both runs, the post-processing was done in a local machine but could have been performed in the Grid server itself, as a post-processing task. Each of the collected result files was in fact a compressed folder (.tgz) comprising log files (with the segmentation errors) and a series of intermediate results and shape model instances, intended to track for events and visualization.

4.3 Quantitative Assessment

The performance assessment analysis was performed using the model state (i.e. position, orientation, scale and shape) from the last matching iteration of the fitting algorithm. Errors were measured by computing the mean unsigned P2S distances from regularly sampled points on the manually segmented contours to the surfaces of the fitted shapes. Two patient data sets were discarded from the assessment because their automatic segmentations were not comparable to the quality of the rest (for all models). The uncorrected field inhomogeneity in one case and a huge pericarditis on the other, confounded the algorithm for any combination of the tuned parameters. Table 2 presents the automatic segmentation results, distinguishing between the shape model subparts (*LV-endo* and *LV-epi*). Values correspond to the shape model with best perfor-

Table 2. Mean $\pm$ SD of the unsigned point to surface (P2S) errors in millimeters. Between brackets, the percentage of improvement achieved by the optimization with respect to previously achieved segmentation accuracy using *ad hoc* settings

Phase	***LV-endo***	***LV-epi***
ED	1.98$\pm$0.54 (27.7%)	1.91$\pm$0.54 (22.0%)
ES	3.6$\pm$1.09 (13.0%)	2.76$\pm$0.89 (16.6%)
Fit of shape model	0.80$\pm$0.13	0.82$\pm$0.16



(a) (b)

Fig. 3. 3D-ASM segmentation. *LV-endo* (a) and *LV-epi* (b) final states. The shape rendered in wireframe corresponds to the fitted shape and the surfaced shape, to the manual one

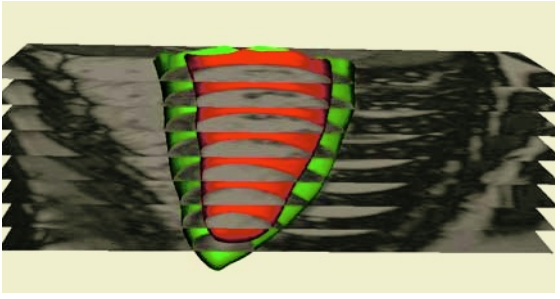


Fig. 4. View across slices of the fitted shape

mance (in fact, the autolandmarked shape model presented in [6]). It is also indicated between brackets the percentage of improvement achieved by the optimization with respect to previously achieved segmentation accuracy using *ad hoc* settings. The third row corresponds to the error that the shape model would have for reconstructing the shapes (either ED or ES) if their exact positions were known. The automatic results are quite within clinically acceptable margins. Fig. 3 shows an example of the fitted model subparts. The manual shape is built from the manual contours and rendered as a surface. Fig. 4 shows a typical result of the achieved segmentation.

5 Conclusions

The presented work serves as an example of Grid computing in a conventional field of software applications, like our research in cardiac image analysis. We believe that in the near future they will become ubiquitous in other biomedical imaging related fields like visualization, processing and archiving, were the availability of huge amounts of data highly requires distributed processing and rapid accessing, as well as sharing computer resources. Surgery planning and simulation or building application oriented morphometric atlases may serve as examples. Based on the experienced benefits with the use of Grid computing during the last two years, and foreseeing the advent of large-scale tests and applications, we strongly see its use as an enabling approach not only for new applications, but for enhancing past developments.

Two distributed applications were presented. The first one made use of the Grid service API, as distributed and non-distributed steps where needed, as well as subprocesses iterations and post-processing. The second Grid-enabled approach was an exhaustive search of the optimal set of parameters for a 3D model-based algorithm for cardiac MRI segmentation. The set-up for distributing such a big number of tasks was a matter of minutes, using the Grid service web portal. This approach is specially suited for Monte Carlo simulations. In both cases, the system took advantage of each resource depending on its usage, and not interfering with end users. When one of the computers in the Grid was not available, its tasks were automatically reassigned. Finally, the system collected the local output from all the units and made them available for download.

In conclusion, we believe that Grid technology solutions are quite valuable as they considerably shorten execution times of exhaustive searches and large-scale image pro-

cessing, effectively enabling the shearing of computing resources between institutions. Our Grid service end user interface allowed for setting-up applications without the need of mastering the topic. In our daily practise, Grid computing have become a real necessity.

References

1. T.F. Cootes, C.J. Taylor, D.H. Cooper, and J. Graham, "Active shape models - their training and application," *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 3859, 1995.
2. T.F. Cootes, G.J. Edwards, and C.J. Taylor, "Active appearance models," *Proc. European Conf. on Computer Vision*, vol. 2, pp. 484498, 1998.
3. S.C. Mitchell, J.G. Bosch, B.P.F. Lelieveldt, R.J. van der Geest, J.H.C Reiber, and M. Sonka, "3D Active Appearance Models: Segmentation of cardiac MR and ultrasound images.," *IEEE Trans Med Imaging*, vol. 21, no. 9, pp. 11671179, 2002.
4. M. B. Stegmann, *Generative Interpretation of Medical Images*, Ph.D. thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, apr 2004.
5. H.C. van Assen, M.G. Danilouchkine, F. Behloul, H.J. Lamb, R.J. van der Geest, J.H.C. Reiber, and B.P.F. Lelieveldt, "Cardiac LV segmentation using a 3D active shape model driven by fuzzy inference," Montreal, CA, Nov. 2003, vol. 2876 of *Lect Notes Comp Science*, pp. 533540, Springer Verlag.
6. A.F. Frangi, D. Rueckert, J.A. Schnabel, and W.J. Niessen, "Automatic construction of multiple-object three-dimensional statistical shape models: Application to cardiac modeling," *IEEE Trans Med Imaging*, vol. 21, no. 9, pp. 11511166, 2002.
7. S. Ordas, L. Boisrobert, M. Bossa, M. Laucelli, M. Huguet, S. Olmos, and A.F. Frangi, "Grid-enabled automatic construction of a two-chamber cardiac PDM from a large database of dynamic 3D shapes," in *IEEE International Symposium of Biomedical Imaging*, 2004, pp. 416419.
8. H.C. van Assen, M.G. Danilouchkine, M.S. Dirksen, J.H.C. Rieber, and B.P.F. Lelieveldt, "A 3D-ASM driven by fuzzy inference: Application to cardiac CT and MR," *IEEE Trans Med Imaging*, 2005, submitted.
9. T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions of Systems, Man and Cybernetics*, vol. 1, no. 15, pp. 116 132, 1985.
10. P.J. Besl and N.D. McKay, "A method for registration of 3D shapes," *IEEE Trans Pattern Anal Machine Intell*, vol. 14, no. 2, pp. 23955, Feb. 1992.

A Grid Molecular Simulator for E-Science

Osvaldo Gervasi[1], Cristian Dittamo[1], and Antonio Laganà[2]

[1] Department of Mathematics and Computer Science, University of Perugia,
via Vanvitelli, 1, I-06124 Perugia, Italy
ogervasi@computer.org
cridit@woodie.unipg.it
[2] Department of Chemistry, University of Perugia,
via Elce di Sotto, 8, I-06123 Perugia, Italy
lag@unipg.it

Abstract. The implementation of **GEMS**, a Grid-based molecular simulator, on the EGEE Grid environment is presented. We discuss the main issues related to the porting of the application on the EGEE Grid platform and the creation of the VO CompChem for the community of Computational Chemists. The real-time visualization of some reaction's magnitudes on Virtual Monitors is also discussed.

1 Introduction

In the present paper we describe the progress made in implementing a Grid based Molecular Simulator and porting the related computational procedure on the EGEE[1] European Grid infrastructure[1].

The EGEE infrastructure has made available to the partner laboratories a massive amount of computational resources. As a result it has become possible within the EGEE project to implement Grid aware applications. This opportunity has been offered through the activities of the EGEE working group *NA4 activity: Application Identification and Support*, not only to high energy physics and bioinformatics communities (as stated in the project from the very beginning) but also to other communities which were ready to implement a complex computational application on the Grid.

The Community of Chemists and Molecular Scientists has developed for this purpose a prototype Grid based European Molecular Simulator (GEMS). **GEMS** has been first developed as part of the **SIMBEX** project[2] under the Action D23 [3] of the COST[4] (*Cooperation in the field of Scientific and Technical Research*) in Chemistry initiative of the European Union.

[1] EGEE is a project funded by the European Union under contract INFSO-RI-508833 The EGEE infrastructure is built on the EU Research Network GEANT. The infrastructure provides interoperability with other Grids around the globe, including Israel, Russia, the US and Asia, with the purpose of establishing a worldwide Grid infrastructure.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 16–22, 2005.

The implementation of **GEMS** on EGEE benefits also of the work done by the workpackage 13 of the italian project *Enabling platforms for high performance computational Grids oriented to scalable virtual organizations*[5] funded by the Italian Ministry of the Education (MIUR).

To continue the development of **GEMS** within NA4, a virtual organization (VO) of Computational Chemists (**CompChem**) has been established and is being populated with researchers of various European countries.

The paper illustrates in Section 2 the structure of GEMS and the steps undertaken to implement the EGEE environment, in section 3 some computational aspects and the results.

2 GEMS

The general structure of **GEMS** is made of four modules: **INTERACTION**, **DYNAMICS**, **COLLECTIVE–MOTIONS** and **OBSERVABLES** each of which takes care of different aspects of the calculation.

INTERACTION: This module is devoted to the determination of the potential energy surface (PES) on which the nuclear motion takes place. This usually implies the integration of the fixed nuclei Schrödinger equation for the electrons at a proper extended grid of molecular geometries. To this end whenever possible, a suite of programs that calculates the value of the electronic energy of molecular systems is used. The calculated values of the electronic energy are then fitted using a procedure based on functional representations of the interactions of polynomial nature either in physical or in bond-order (BO) coordinates. This procedure builds a Fortran routine **Pesfunction**.

When *ab initio* values are neither available from a data-bank nor obtainable by an ad hoc calculation, empirical approaches are often adopted by calling a force field procedure. This procedure builds an estimate of the interaction by summing simple two, three and four body functionals containing empirically determined parameters [6]. This procedure can also be bypassed when a real time ("on the fly") evaluation of the electronic energy is performed at each step of the dynamical calculation. Alternatively, when the **Pesfunction** routine is already locally available it is imported and properly interfaced to the next section **DYNAMICS**.

DYNAMICS: In this second module dynamical calculations can be carried out using either Quasiclassical Trajectory or Quantum Mechanical (QM) approaches. For small systems (a few atoms) QM calculations can be performed. In this case either a time dependent or a time independent reactive scattering program is activated according to whether a single initial state (for an interval of energy values) or a fixed energy (for all the open states) calculations needs to be performed.

For higher energies or heavier systems a reduction of the complexity of the calculations may be desirable. In this case a trajectory approach is adopted and the related procedure is articulated into several programs. The program dealing

with atom diatom systems is **ABCtraj** and the program dealing with few body diatomic system is **VENUS**.

COLLECTIVE–MOTIONS: The third module takes care of large systems using either dynamical or statistical means. Dynamical methods treat the system as an ensemble of individual particles. In this case collective motions are dealt as a sum of individual particle motions. More often other approaches accounting for the different qualitative nature of the subsystems involved in electrodynamics, fluid dynamics, interphase processes, etc, are adopted by building into the equations statistical considerations. This module is no further detailed here since in the implemented version of **GEMS** it has not been included. However in the collaborations of the participating laboratories the study of plasmas, non equilibrium processes, shock waves (still within the domain of the gas phase) will soon be considered.

OBSERVABLES: This fourth module is devoted to the reconstruction and visualization of the observable properties and of other features relevant to experimental measurements or practical applications. To this end the outcomes of the **DYNAMICS** module are postprocessed by a procedure which either reconstructs the signal measured by the beam apparatus or estimates other quantities that the user may wish to know regardless of the fact that one deals with real observables. These observables or pseudo–observables are extremely useful for rationalizing the system behaviour. For this reason one has to pay significant attention to the rendering of the results and to the relevant "virtual monitors".

The implemented version of the **GEMS** makes use of a Web Workflow environment based on several server-side scripts written in PHP that are computed by the Web Server installed on the User Interface (**UI**) host. Using the Workflow environment the user chooses the operating conditions of the simulation (i.e: the chemical system, the Potential Energy Surface, the initial conditions of the chemical species, etc) and submits to the Grid the application running the simulation. The input data selected by the user are passed to the execution environment. The user can monitor the simulation by launching some Java applets on the client side that will communicate with the Web server running on the **UI** for getting updated results.

The Web Server performs the following tasks:

- receives the configuration values entered by the user through the web;
- builds the necessary files scripts and files with simulation data storing them in specific path on the **UI** filesystem;
- executes shell scripts to run the parallel MPI job on the Grid.

A shell script launches a process, named **SimGate**[7], receives updated results from the parallel job and sends them to the client's Java applets during the execution of the simulation. For launching the parallel job on the Grid a **JDL** file is built from the Web environment and sent to the Resource Broker of the Grid environment.

3 Computational Issues and Results

The Simulator has been ported on the Grid platform of EGEE by implementing it on the cluster of the Department of Chemistry of the University of Perugia. This cluster has 14 double-processor nodes which are interconnected through a dedicated Gigabit Ethernet switched network.

The test has been run on two applications, **ABCtraj** (for atom-diatom reactions) and **VENUS** (for multibody reactions). These computational engines run concurrently using the Message Passing Interface (MPI) [8] libraries and adopting a Farmer-Worker parallel programming paradigm, based on SPMD (Single Program Multiple Data) pattern. The workload of trajectory subsets is dynamically assigned to the nodes which also collect partial results.

The whole algorithm is illustrated in figure 1.

The input data file read by the Farmer program is created by the latest web page of the work-flow environment, designed to help a user in defining

The ABCtraj algorithm

Farmer (process with `MPI_ID = 0`):

Read input data from file `/tmp/simbex_`*job_id*`.input`;
Initialization of global variables;
Calculate a seed for each trajectory to be computed;
Send to each worker the trajectory number and the corresponding seed;
while(there are some trajectories not computed)
    Wait for receiving relevant data from *worker*;
    Send to *worker* new work unit;
    Write updated info into output file for monitoring;
end while
Send an 'end of work' to worker;
Carry out the remaining calculations relevant to the evaluation;
Write rate coefficients, cross sections and product distributions for the different reaction channels into output file for monitoring;

Worker:

while('end of work')
Receive assigned trajectory number and random number seed;
Generate in the string of pseudorandom numbers needed to set the initial conditions of the related trajectory;
Integrate the trajectory;
Send the results to Farmer;
Wait for a new work unit;
end while

Fig. 1. Logical steps implemented in ABCtraj program

the input parameters of the simulation through the Graphical User Interfaces (GUI) implemented in the PHP scripting language. For executing this server-side scripts the Apache Web Server was chosen. In this way any authenticated user can launch a simulation through a web browser.

To execute a simulation one has to issue the following steps:

1. **connect** to the GEMS Workflow environment[9]
2. **authenticate** or register (the first time only);
3. **configure** the input parameters of the simulation;
4. **launch** the simulation on the Grid (transparent to user);
5. **monitor** the application through Virtual Monitors customized for each computational engine.

In the fourth step the latest web page gathers all info and writes it into an input data file that will be read by the Farmer running on the Grid infrastructure. The interactivity of the simulation is driven by **SimGate**, a daemon launched by the Web Server on the **UI** as soon as the user starts the simulation on the Grid. It manages the exchange of information between the Virtual Monitors and the simulation job running on the Grid. The communication pattern is client-server. The Virtual Monitors, running on the client side, asks for updated values to **SimGate** which answers by supplying the updated values. Concurrently **SimGate** receives updated values from the Farmer. The **SimGate** protocol has been designed to solve the problems related to the management of multiple connections of concurrent simulations. In the fifth step a user can monitor the evolution of the simulation through Virtual Monitors implemented in Java.

For illustrative purposes the atom diatom case study $H + ICl$ is discussed here[10]. It can lead either to the HI or to the HCl product. For this atom-diatom system the routine **Pesfunction** is already available and the trajectory application **ABCtraj** was used.

After having inputed the operating conditions of a given simulation, the user can choose the relevant Virtual Monitors by selecting them in the last page of the Workflow environment. Two screenshots of the Opacity Function (the fraction of

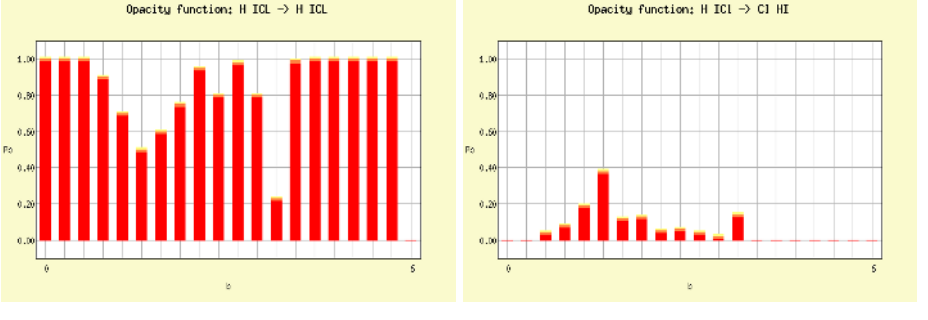


Fig. 2. Example of the Opacity Function Virtual Monitors for the $H + ICl \rightarrow H + ICl$ (lhs panel), the $H + ICl \rightarrow HI + Cl$

trajectories started at a given value of the impact parameter b plotted as a function of b expressed in angstrom) obtained from the $HICl$ run are shown in fig. 2.

A useful feature of the application is that the user can revisit the Virtual Monitors of previously performed simulations. This function is enabled when storing the final results of the simulation in a user's directory. When a user wants to see an archived simulation she/he can select the wanted one from a summary table shown in a dedicated web page.

The tests made showed a non negligible time of activation of the Grid during the initial scheduling of the job (due to the time necessary to perform the required authentications through the use of Grid certificates). Once the job is started on the Grid, however, the simulation shows a good response evidencing that even for systems described by a PES easy to compute, the largest fraction of computing time goes into the integration of the distributed trajectories. The Grid demonstrates even better its power when systems with a PES requiring large amount of computing power are investigated. In this case the possibility of running simultaneously a large number of tasks on the Grid dramatically reduces the time required to complete the simulation making a clear advantage over a single parallel machine even if with a large number of nodes.

4 Conclusions

In the present paper we have described the efforts spent by the Chemists and Molecular science community in establishing a virtual organization within the EGEE project. This first example is based on the implementation of a gas phase molecular simulator derived from previous virtual laboratories international experiences of the chemistry community. The simulator has shown to fully exploit the Grid features by running the application in parallel using MPI and by allowing the interactive monitoring of the simulations through virtual monitors made by Java applets. Even if the response time of the Grid makes the simulations involving fast trajectories and a small number of events not so advantageous, the Grid demonstrates its power for molecular systems requiring a large number of slow trajectories, because of the high number of working nodes involved.

5 Acknowledgment

Thanks are due to and COST and MIUR FIRB Grid.it (project RBNE01KNFP on High Performance Grid Platforms and Tools) for financial support.

References

1. `http://www.eu-egee.org`
2. COST Action N.D23, 003/2001, *SIMBEX: a Metalaboratory for the a priori Simulation of Crossed Molecular Beam Experiments* (duration: from 1/2/2001 to 31/1/2006), Coordinator: O. Gervasi, Dept. of Mathematics and Informatics, University of Perugia (2001).

3. Laganà, A.: METACHEM: Metalaboratories for cooperative innovative computational chemical applications, METACHEM workshop, Brussels, November (1999) (see also [2]);
`http://costchemistry.epfl.ch/docs/D23/d23-main.htm`
4. `http://www.cordis.lu/cost/home.html`
5. `http://www.grid.it`
6. Burkert, U., Allinger, N.L.: Molecular Mechanics ACS Monograph Series, Washington DC, USA, (1982).
7. Gervasi, O., Laganà, A.: **SIMBEX**: a portal for the a priori simulation of crossed beam experiments, Future Generation Computer Systems, 20, 703-715 (2004)
8. Message Passing Interface Forum, Int. J. of Supercomputer Applications 8(3/4), 1994; Smir, M., Otto, S., Huss-Ledermam, S., Walker, D., Dongarra, J.: *MPI: The complete reference*, MIT Press (1996).
9. `http://gems.simbex.org`;
10. Gervasi, O., Laganà, A., Lobbiani, M.: Toward a GRID based Portal for an a Priori Molecular Simulation of Chemical Reactivity, Lecture Notes in Computer Science 2331, pp. 956-967 (2002)

Application Driven Grid Developments in the OpenMolGRID Project

Bernd Schuller, Mathilde Romberg, and Lidia Kirtchakova

Research Center Jülich, Central Institute for Applied Mathematics, Jülich, Germany
{b.schuller, m.romberg, l.kirtchakova}@fz-juelich.de

Abstract. Within the frame of the OpenMolGRID project, several extensions to the underlying UNICORE Grid infrastructure have been developed. This article gives an overview of these developments, focussing on the support for complex scientific workflows and a newly developed command line client for UNICORE.

1 Introduction

The OpenMolGRID (Open Computing Grid for Molecular Science and Engineering) project[1] [1] deals with the integration of data, applications and the modelling of molecular design workflows in a Grid environment. The overall aim of the project is to provide an extensible, information rich environment for molecular engineering and molecular design, using resources available on the Grid in a seamless and secure fashion. The underlying scientific methodology, quantitative structure–activity relationship (QSAR), and other application-specific aspects of the project have been described in detail in [2] and [3]. From an information technology perspective, the unique challenges of the project, which make it an excellent application domain and testing ground for Grid techniques, can be summarised as follows:

- Molecular design and engineering is computationally very intensive, and can generate huge amounts of data;
- Data from diverse sources needs to be integrated using data warehousing techniques and made accessible seamlessly;
- The scientific processes, or workflows, typically involve diverse and heterogeneous data and compute resources;
- The workflows are fairly complex, involving multiple, dependent computational steps

As the Grid middleware underlying the OpenMolGRID system, UNICORE was chosen. The challenges summarised above resulted in the development of several new middleware components, that are generic enough to be used in other contexts as well.

[1] OpenMolGRID is funded in part by the EC under contract IST-2001-37238.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 23–29, 2005.

The remainder of this paper is organised as follows. After introducing the UNICORE middleware we describe the components we have developed for workflow support and the new command line interface for UNICORE.

2 Unicore

The OpenMolGRID system is based on UNICORE[2] (UNiform Interface to COmputer REsources) [4] as the underlying Grid middleware. UNICORE can be characterised as a vertically integrated Grid system, that comprises a graphical client and various server and target system components. UNICORE continues to be developed in the recently started EU FP6 project UniGrids [5], and is being used in various projects such as DEISA [6]. It is available fully open-source under a BSD-type license from the SourceForge repository [7].

For use within an application-centric project such as OpenMolGRID, the main asset of UNICORE is its excellent support for legacy software. UNICORE allows applications to be integrated into the Grid easily [8]. On the server side, the software can be installed "as-is". The client's graphical user interface is extensible by application specific user interface components (plugins).

Furthermore, UNICORE offers strong, end-to-end security through the use of an X.509 public key infrastructure. Security is an important requirement of the pharmaceutical industry, which is one of the key targeted user communities of the OpenMolGRID project.

UNICORE jobs are represented by a Java object, the Abstract Job Object (AJO). It is important to note that in order to be executable, such an AJO must contain all the information about Grid sites and resources. The user can create AJOs easily and comfortably using the graphical UNICORE client, but she needs to allocate resources, select the correct sites for the individual subtasks, and take care of data transfers between sites that may be needed. Obviously, in the case of a large, or rapidly changing Grid, this gets increasingly tedious and difficult.

Within OpenMolGRID, new components have been developed for making the resource selection and allocation procedures much easier, so the users can concentrate on their scientific applications instead of having to deal with the tedious details.

3 OpenMolGRID Workflow Support Architecture

While OpenMolGRID adds significant functionality to the basic UNICORE software, it has been an important design principle to leave the basic UNICORE software untouched, if possible. Thus, all application plugins and server-side

[2] UNICORE has been developed in several research projects partly funded by the German Ministry of Education and Research under Contracts 01-IR-703 (UNICORE) and 01-IR-001 (UNICORE Plus) and by EC under Contract IST-1999-20247 (EUROGRID) and IST-2001-32257 (GRIP).

tools developed within OpenMolGRID are usable individually, and are fully functional when used outside the OpenMolGRID system, i.e. in a standard UNICORE context. Also, the system has been designed for maximum extensibility and flexibility, to be able to easily accomodate future changes and extensions in the OpenMolGRID system as well as changes in the underlying UNICORE software.

OpenMolGRID workflow support consists of a metadata layer for applications, an extension of the client side plugin interface and several components that deal with resource management and workflow support.

3.1 Application Metadata

As is usual in UNICORE, integrating a software package into the Grid system is done by installing the software executable on a Grid node (a virtual site, or *Vsite* in UNICORE's terminology), and by providing a metadata file that is associated with the application. Additionally, an application-specific Client extension (a *plugin*) providing a graphical user interface should be provided.

In OpenMolGRID, the *application metadata* play an important role. They are used to define the services provided by the application, analogous to a WSDL document in the web services world. Each such service, called "task" within the OpenMolGRID system, is defined by the following elements:

- the task name
- input file(s) specification
- output file specification
- options

Input and output files are always associated with a high-level datatype, which can be compared to a MIME type. This allows matching input and output between subsequent steps in a complex workflow.

This metadata information is part of the server side resources (similar to, for example, the number of CPUs the site provides) and is sent to the Client. There, the metadata are used extensively by the workflow support components, as described in section 3.3 below.

3.2 Client-Side Plugin Extension

On the Client side, plugins must implement a new interface, that allows the workflow support components to set up the job without user intervention. This interface has a function that returns the supported *tasks*, so that the system can match client side plugins and server side applications easily. Furthermore, the interface includes functions to set and query input and output filenames, and to set options.

3.3 Workflow Support Components

Using the application integration layer and the additional plugin interface described above, support for higher-level workflows could be introduced. Instead of

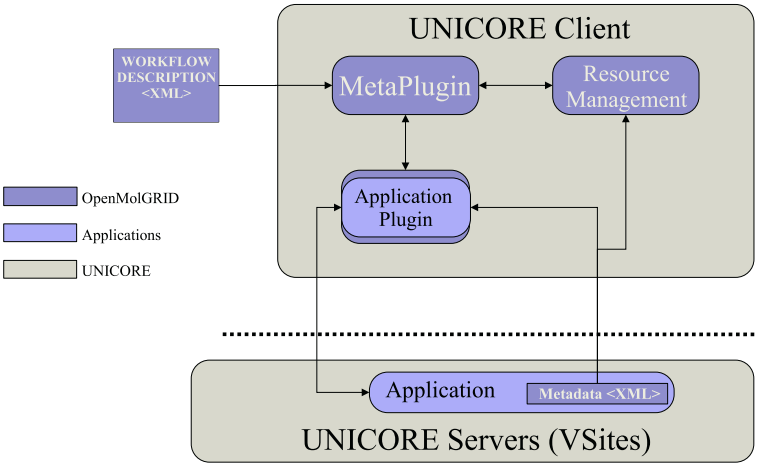


Fig. 1. Workflow support architecture

preparing an executable UNICORE job with the usual methods, an XML workflow description is read by the workflow support components developed within OpenMolGRID. From this XML workflow, a UNICORE job is then generated automatically.

Compared to the usual UNICORE way of creating a job, this has a number of advantages. The workflows are fairly high-level, focussing on tasks and dependencies, and thus are very close to the actual scientific workflows the users are interested in. The user need not specify where a given task will be executed, nor need she worry about where certain software is available. The system will take care of mapping the workflow to an executable UNICORE job. The system will also take care of inserting data transfer tasks between Grid nodes, if needed.

The extensions to UNICORE which make up the OpenMolGRID workflow support are given schematically in Fig. 1. The main new component is the MetaPlugin, that deals with setting up a ready-to-submit UNICORE job from an XML workflow description. It is supported by a resource management plugin that reads the server-side application metadata and checks the availability of server-side applications and client-side plugins.

Apart from the basic functionality of setting up the job, the MetaPlugin has sophisticated functionality that allows "distributing" data-parallel tasks to run on multiple Grid nodes. Its design is highly modular and offers various interfaces for later enhancements. For example, currently resource allocation does not take dynamic parameters such as system load into account. However, more advanced brokering functions can be added easily.

3.4 XML Workflow Description

The workflows that play a role in OpenMolGRID are high-level workflows, focussing on *tasks* and their dependencies. Since none of the available workflow de-

scription schemas fully matched our needs, we developed a simple XML schema ourselves. As an example, a typical workflow to compute molecular descriptors for a molecular structure starting from two-dimensional chemical structures can be expressed as:

- convert to the three-dimensional molecular structure
- optimize the molecular geometry
- compute molecular descriptors

In our workflow schema, this workflow would, in its simplest form, just state the tasks and dependencies and would look as follows

```
<?xml version="1.0"?>
<workflow>
<task name="2Dto3DConversion" identifier="convert" id="1"/>
<task name="SemiempiricalCalculation" identifier="optimize"
id="2"/>
<task name="DescriptorCalculation"
identifier="Calculate descriptors" id="3"/>
<dependency pred="1" succ="2"/>
<dependency pred="2" succ="3"/>
</workflow>
```

However, this workflow still needs some user input: the initial input data need to be setup, computational parameters must be chosen, the output data needs to be stored permanently. UNICORE might need some resource requests, for example CPU time allocation on resource management systems. The user might wish to parallelize a task on as many grid nodes as possible.

All these things can be done automatically. The workflow schema includes support for

- grouping tasks;
- flow control, such as repeat loops and an if–then–else construct;
- using local data as input;
- requesting specific resources (for example, a certain data source);
- requesting for a task to be run on multiple sites (if the task is data-parallel and the input data can be split in multiple parts);
- setting options for a task;
- allocating resources (such as CPU time) to a task.

In effect, these workflows can be used very flexibly. They are high-level enough so the user need not specify every tiny detail. But, if needed or wanted by the user, they can include more detailed information about Grid nodes and resource requirements.

4 Command-Line Toolkit

UNICORE lacks a powerful command-line toolkit, or a simple API that could be used by programmers or application developers to make use of Grid resources from within their applications. Furthermore, for convenient automatisation of processes (for example, batch processing) the standard, graphical UNICORE client is not convenient. To address these issues, we have developed a powerful command-line interface to UNICORE.

This tool offers an AJO generation function that builds a UNICORE job dynamically from an XML workflow description. Furthermore, functions to run jobs, monitor them, and fetch the job results are provided. The job generation facility is based on the MetaPlugin, and uses the full OpenMolGRID metadata layer. Therefore, the same high-level workflows can be run in the GUI client and in the command-line client.

On top of the command-line client, a simple front-end has been developed that simplifies running workflows through the command-line client. A workflow can be submitted to the command-line client by placing it in the "request" queue. From this, a UNICORE job will be built and run, and ultimately the result files can be found in the result directory. This toolkit thus offers very powerful batch-processing capabilities.

The command-line toolkit is already used successfully in OpenMolGRID's data warehousing component[3]. When loading data into the data warehouse, Grid resources are used to perform conversion tasks and computatation of supplementary data. In this fashion, the subsequent data mining steps are simplified, as standard computations have already been performed by the data warehousing components.

Specifically, for each chemical structure that is loaded into the warehouse, the following data transformations are performed. The structure is converted to a three-dimensional representation, which is then optimised. Finally, molecular descriptors are calculated, and all this data is stored in the warehouse. This process is fully automated by running the appropriate XML workflows through the command-line client. Because the data transformations are done on a per structure basis, the available Grid resources are used very efficiently, as the Grid sites are used in a round-robin fashion automatically by the MetaPlugin.

5 Summary

Within OpenMolGRID, we have developed a number of powerful tools for the UNICORE Grid infrastructure. While these tools were designed with a specific field of application in mind, they are completely generic, and can be used in a variety of contexts. We believe the workflow support tools can help speed up, automatise and standardise scientific processes by integrating data and applications on a UNICORE Grid and offering support for complex workflows. In addition, the command-line client toolkit offers new ways of accessing UNICORE resources that are highly suitable for batch-processing tasks, where no user intervention is needed or wanted.

These new tools are already used successfully in OpenMolGRID, and end users are starting to make use of the new possibilities for doing their science that this system offers.

References

1. The OpenMolGRID project: http://www.openmolgrid.org
2. Mazzatorta P., Benfenati, E., Schuller, B., Romberg, M., McCourt, D., Dubitzky W., Sild, S., Karelson, M., Papp, A., Bagyi, I., Darvas, F.: OpenMolGRID: Molecular Science and Engineering in a Grid Context; in: Proceedings of the PDPTA 2004 Conference, June 21-24, Las Vegas, Nevada, USA
3. Dubitzky, W., McCourt, D., Galushka, M., Romberg, M., Schuller, B. Grid-enabled data warehousing for molecular engineering; Parallel Computing **30** (2004), 1019–1035
4. Romberg, M.: The UNICORE Grid Infrastructure; Scientific Programming Special Issue on Grid Computing **10** (2002) 149–158
5. UniGrids homepage: http://www.unigrids.org
6. DEISA homepage: http://www.deisa.org
7. UNICORE SourceForge pages: http://unicore.sourceforge.net
8. Pytlinski, J., Skorwider, L., Huber, V., and Bala, P.: UNICORE - A uniform platform for chemistry on the Grid; Journal of Computational Methods in Science and Engineering, **2** (2002), 369–376

ATLAS Data Challenge 2: A Massive Monte Carlo Production on the Grid

Santiago González de la Hoz[1], Javier Sánchez[1], Julio Lozano[1], Jose Salt[1], Farida Fassi[1], Luis March[1], D.L. Adams[2], Gilbert Poulard[3], Luc Goossens[3], and DC2 Production TEAM (ATLAS Experiment)[3]

[1] IFIC- Instituto de Física Corpuscular, Edificio de Institutos de Investigación, Apartado de Correos 22085, E-46071 Valencia, Spain
{Santiago.Gonzalez, Javier.Sanchez, Julio.Lozano, Jose.Salt, Luis.March, Farida.Fassi}@ific.uv.es
http://ific.uv.es/

[2] Brookhaven National Laboratory, Upton NY 11973, USA
dladams@bnl.gov
http://www.bnl.gov/

[3] CERN, European Organization for Nuclear Research, 1211 Geneva 23, Switzerland
{Gilbert.Poulard, Luc.Goossens}@cern.ch
http://www.cern.ch/

Abstract. The study and validation of the ATLAS Computing Model started three years ago and will continue for few years in the context of the so-called Data Challenges (DC). DC1 was conducted during 2002-03; the main goals achieved were to set up the simulation data production infrastructure in a real worldwide collaborative effort and to gain experience in exercising an ATLAS wide production model. DC2 (from May until December 2004) is divided into three phases: (i) generate Monte Carlo data using GEANT4 on three different Grid projects: LCG, GRID3 and NorduGrid; (ii) simulate the first pass reconstruction of real data expected in 2007, and (iii) test the Distributed Analysis model. Experience with the use of the system in world-wide DC2 production of ten million events will be presented. We also present how the three Grid flavours are operated. Finally we discuss the first prototypes of Distributed Analysis systems.

1 Introduction

The ATLAS experiment [1] is a large detector for the study of high-energy proton-proton collisions at the Large Hadron Collider (LHC) [2], presently under construction at the European Organization for Nuclear Research (CERN) and scheduled to start operation in 2007. In the ATLAS Computing Model, after reduction of the data by the online trigger processor farms, the expected volume of data recorded for off-line reconstruction and analysis will be of the order of 1 PB (10^{15} bytes) per year. Therefore, in 2002 a series of Data Challenges (DC's) were planned with the purpose of the validation of the Computing Model, of the complete software suite, of the data model, and to ensure the correctness of the technical choices to be made. A major

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 30 – 39, 2005.

feature of the first Data Challenge (DC1) [3] was the development and the deployment of the software required for the production of large event samples required by the High Level Trigger and Physics communities, and the production of those large data samples involving institutions worldwide.

It should be noted that it was not possible to produce all the data at CERN, since the resources to perform this task on a reasonable timescale were not available.

The ATLAS collaboration decided to perform these DC's in the context of the LHC Computing Grid project, LCG [4], to which ATLAS is committed, but also to use both the middleware and the resources of two other Grid projects, GRID3 [5] and NorduGrid [6]. The job of the LCG is to prepare the computing infrastructure for the simulation, processing and analysis of the LHC data for all four LHC collaborations. The LCG scope spans both the common infrastructure of libraries, tools and frameworks required to support the physics application software, and the development and deployment of the computing services needed to store and process the data, providing batch and interactive facilities for the worldwide community of physicists involved in LHC. The main emphasis of the LCG project is the deployment of Grid technologies for the LHC computing. Both GRID3 and NorduGrid have similar approaches using the same foundations (GLOBUS) as LCG but with slightly different middleware.

Concerning the ATLAS data analysis model many important software components remain to be done. They will be based on the long term experience of previous experiments and on the emerging new technological breakthroughs. The development and integration of the detector specific reconstruction and physics analysis software, followed by their deployment to the Grid in large scale Data Challenges, will enable ATLAS to validate its Computing Model and to demonstrate its physics potential.

1.1 Scientific Originality and Innovation

The high particle collision rate and the large event size in ATLAS make the offline computing much more difficult than in previous experiments, even comparing to CDF [7] and D0 [8] (two experiments which are currently running at the Fermilab laboratory in the United States). With respect to these two experiments, the event rate in ATLAS will be a factor of 50 and the event size will be eight times larger.

The offline computing will have to deal with an output event rate of 100 Hz, i.e. 10^9 events per year with an average event size of 1 Mbyte. This means that new algorithms for data reconstruction are needed in order to achieve the required reconstruction latencies and the necessary large reduction of the data volume.

The new Grid technologies will provide the tools to analyze all the data recorded by ATLAS and to generate the large "Monte Carlo" simulation samples required. They are expected to make feasible the creation of a giant computational environment out of a distributed collection of files, databases, computers, scientific instruments and devices.

2 ATLAS Production System

In order to handle the task of ATLAS DC2 an automated production system [9] was designed. All jobs are defined and stored in a central database. A supervisor agent

(Windmill) [10] picks them up, and sends their definition as XML message to various executors, via a Jabber server. Executors are specialised agents, able to convert the XML job description into a Grid specific language (e.g. JDL, job description language, for LCG). Four executors have been developed, for LCG (Lexor) [11], Nordugrid (Dulcinea) [12], GRID3 (Capone) [13] and legacy systems [14], allowing the Data Challenge to be run on different Grids.

When a LCG job is received by Lexor, it builds the corresponding JDL description, creates some scripts for data staging, and sends everything to a dedicated, standard Resource Broker (RB) through a Python module built over the workload management system (WMS) API. The requirements specified in the JDL let the RB choose a site where ATLAS software is present and the requested computing power is available. An extra requirement is a good outbound connectivity, necessary for data staging.

The actual executable is wrapped in a script that performs various tasks: Check the ATLAS software (s/w) installation on the worker nodes (WN); Download and install packages [15] for the required application; Set up the ATLAS s/w environment; Stage-in the input files, perform the transformation and Stage-out the results.

For data management, a central server, Don Quijote (DQ) [16] offers a uniform layer over the different replica catalogues of the 3 Grid flavors. Thus all the copy and registration operations are performed through calls to DQ. The s/w distribution is installed using LCG tools.

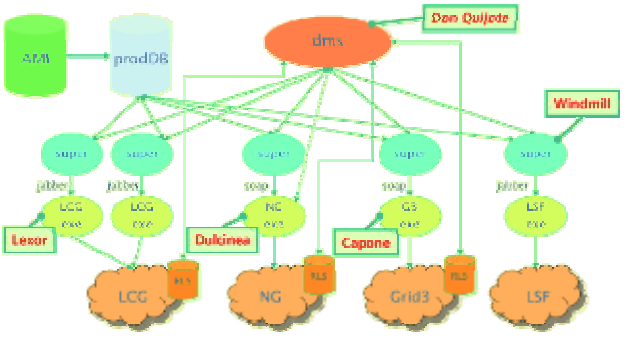


Fig. 1. The ATLAS production system consists of 4 components: The production database; The windmill supervisor; The Executors; Don Quijote, the Atlas Data Management System

3 DC2 Production Phases

During the LHC preparation phase, ATLAS has large needs for simulated data which allow understanding the detector performance. These "Monte Carlo" simulations are done using the full ATLAS chain which runs in the [17] Athena framework [18] and consists of:

- Event generation: (Pythia [19] and Herwig [20] generators), writing out the generated events to ROOT files [21], using POOL persistency [22]. The event size is 60 KB and the computer power required 156 KSI2K-s.

- GEANT4 simulation [23]: reading the generated events via POOL and running GEANT4 simulation, writing out the simulated hits from all ATLAS sub-detectors to ROOT files. The event size is1.9 MB and the computer power required 504 KSI2K-s.
- Digitization: reading in the simulated hits via POOL; writing out the RDO's (Raw Data Objects) to ROOT files. The event size is 1.9 MB and the computer power required 16 KSI2K-s

The relevant output information of the reconstruction will be stored in the form of ESD (Event Summary Data) and in a more compact form, more suitable for analysis, AOD (Analysis Object Data).

The Phase 1 of the ATLAS DC2 started in July 2004 and it is divided into five parts: Event generation and detector simulation; Pile-up and digitization. Pile-up is the superposition of "background" events with the "signal" event and digitization data stores the response of the sensitive elements of the detector. The output, called byte stream data, looks like detector "Raw Data"; Data transfer to CERN (~35 TB in 4 weeks); Event mixing, events from different physics channels are "mixed" in "ad-hoc" proportions. For the Pile-up, the event size is 3.3 MB and the computer power required 144 KSI2K-s and for the Event mixing 3 MB and 5400 SI2K-s.

ATLAS is currently using 3 Grid flavors LCG [4], GRID3 [5] and NorduGrid [6] in different development states. The ATLAS DC2 collaboration finished the simulation part at the end of September 2004. 10 million events were generated and simulated using the three flavors. The contribution from each flavor is shown in Figure 2.

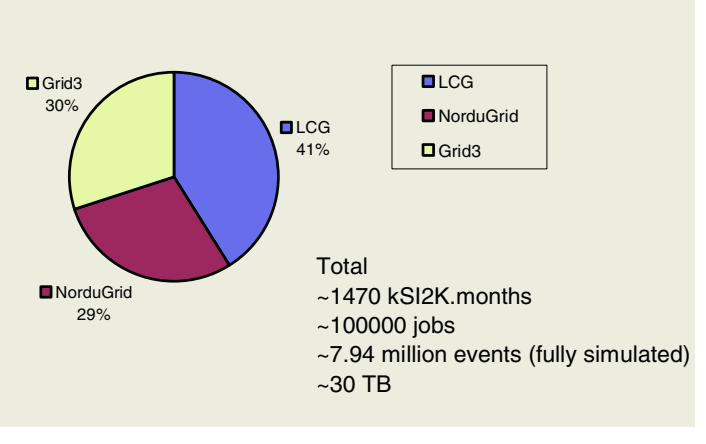


Fig. 2. The chart plots the contribution of each Grid flavor in the generation and simulation parts for the ATLAS Data Challenge

3.1 DC2 Production Using GRID3

The GRID3 collaboration has deployed an international Data Grid. The facility is operated jointly by the U.S. Grid projects iVDGL, GriPhyN, PPDG, and the U.S. participants in the LHC experiments.

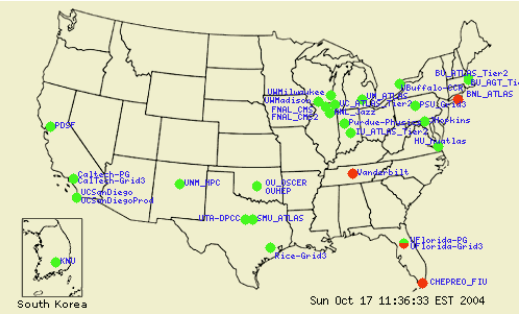


Fig. 3. Geographical distribution of GRID3

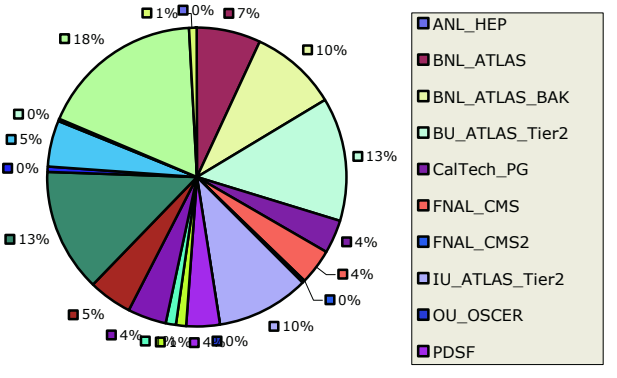


Fig. 4. Jobs contribution by site in GRID3 for the simulation part in DC2

The deployed infrastructure (see figure 3) has been in operation since November 2003 involving 27 sites, a peak of 2800 processors, work loads from 10 different applications exceeding 1300 simultaneous jobs, and data transfers among sites greater than 2 TB/day.

Figure 4 shows the jobs contribution to the ATLAS DC2 in the simulation part. A specific production system has submitted jobs to GRID3 sites at full scale using their shared facilities. Around 30000 jobs were finished successfully, 2.4 million of events and 8 TB were produced, and more than 0.5 million CPU-hours were consumed.

3.2 DC2 Production Using NorduGrid

The NorduGrid project is established mainly across Nordic countries but includes sites from other countries. It was the first to reach production quality level and contributed to a significant part of the DC1 production. It provides resources for DC2 and support for production on non-RedHat 7.3 platforms (e.g. Mandrake 8.0, Debian 3.0).

The NorduGrid resources range (see figure 5) from the original small test-clusters at the different physics-institutions to some of the biggest supercomputer clusters in

Scandinavia. It is one of the largest operational Grids in the world with approximately 4000 CPU's, storage capacity of 14 TB, involving 40 sites and 11 countries.

Figure 6 shows the jobs contribution to the ATLAS DC2 in the generation and simulation part. Around 30000 jobs were finished and 2.4 million of events were produced.

Fig. 5. Geographical distribution of NorduGrid

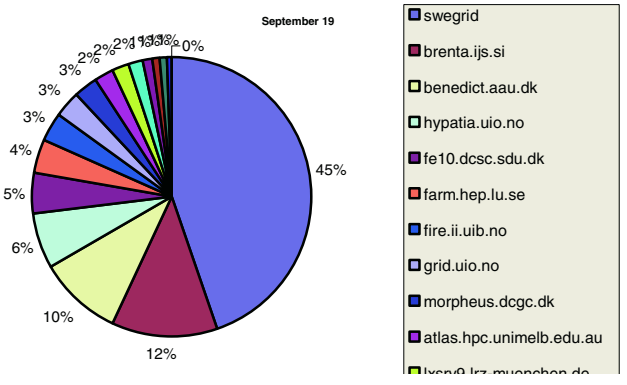


Fig. 6. Jobs contribution by site in NorduGrid for the simulation part in DC2

3.3 DC2 Production Using LCG

The LCG project is built upon the most stable developments of the European Data Grid middleware (EDG) [24], the US Virtual Data Toolkit project (Globus) [25] and European DataTag [26] monitoring tools.

The requirements for LHC data handling are very large, in terms of computational power, data storage capacity and data access performance. It is not considered feasi-

ble to fund all of the resources at one site, and so it has been agreed that the LHC computing services will be implemented as a geographical distributed Computational Data GRID (see figure 7). This means that each service is using computing resources, both computational and storage, installed at a large number of Regional Computing Centres in many different countries, interconnected by fast networks. The deployed infrastructure has been operating since 2003 with 82 sites of 22 countries at peak of 7269 processors and a total storage capacity of 6558 TB.

Figure 8 shows the LCG jobs distribution in the generation and simulation part. LCG sites ran production systems at scale during this period using shared facilities. Around 40000 jobs were finished successfully and 3.1 million of events were produced.

Fig. 7. Geographical distribution of LCG

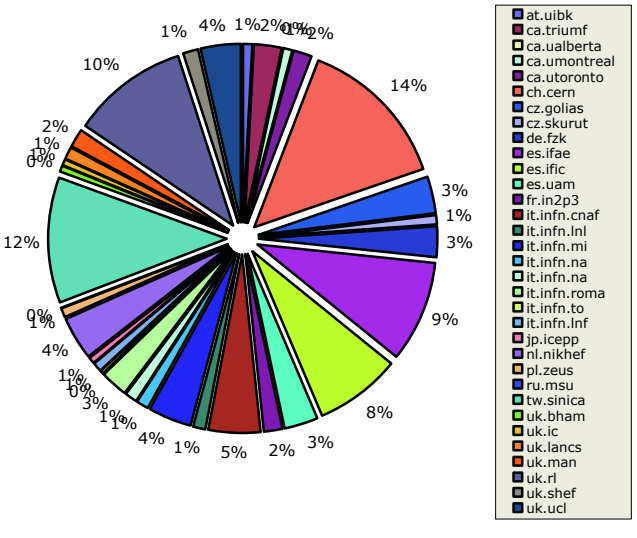


Fig. 8. Jobs contribution by site in LCG for the simulation part in DC2

4 Experiences

By design, the production system was highly dependent on the services of the Grids it interfaces to. The beta status of the implementation of these services has caused troubles while operating the system. For example, the Globus Replica Location Services (RLS) [27], the Resource Broker and the information system were unstable at the initial phase. But it was not only the Grid software that needed many bug fixes, another common failure was the mis-configuration of sites.

On the other hand, since the beginning of DC2 to the end of September around 6 TB of data have been moved using Don Quijote servers. The automatic production system has submitted about 235000 jobs belonging to 158000 job definitions in the Database, producing around 250000 logical files and reaching approximately 2500-3500 jobs per day distributed over the three Grid flavors.

5 Distributed Analysis System

The Grid infrastructure will be used also by the physicists to perform the analysis of the reconstructed data. The ADA [28] (ATLAS Distributed Analysis) project aims at identifying those software components which will allow the end-user to take benefit from the Grid. The goals are rather challenging since they should cover a wide variety of applications involving production and analysis of large distributed data samples. It is possible to identify a generic way in which the user will interact with the Grid resources: she will define an input dataset which will be processed by an application whose behavior will depend on a set of parameters and/or user's code and it will be transformed into an output dataset. Those elements will characterize the job which will be sent to a high level service in charge of the job submission and output management.

The aforementioned software components should provide a way to implement the different processing elements: input dataset, application, task (user's source code and parameters), and output dataset and submission service. All these components make up the Analysis Job Description Language (AJDL) [29]. The submission component involves different tasks including the localization of resources, the splitting of the jobs, the staging of input data and the recollection of the output data.

The ADA architecture is sketched in figure 9. Command line interfaces allow the user to interact via AJDL with the analysis services (AS) which give access to distributed computing power.

DIAL [30] (Distributed Interactive Analysis of Large datasets) is a project within ADA which strives to demonstrate the feasibility of distributed analysis. It implements the AJDL components in C++ classes which can be interfaced through the ROOT [21] analysis framework or the Python [31] command line interface.

The ATPROD (ATLAS PRODuction) system has been used for the DC2 ATLAS data production. Work has started to build an interface allowing users to make their own small scale productions.

ARDA [32] (A Realization of Distributed Analysis for LHC) is another project to deliver a prototype for distributed analysis. The underlying middleware is based on the EGEE [33] gLite [34] package. User interfaces are at this moment under development.

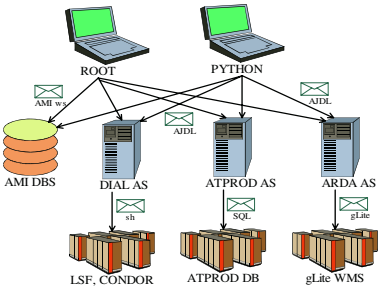


Fig. 9. Main current elements of the ADA schema, including services and interfaces

Currently, a fully operational system based on the DIAL analysis service is available; it accesses a Condor [35] based PC cluster for long running jobs and a LSF [36] based cluster providing interactive response, both located at BNL (Brookhaven National Laboratory). A prototype analysis service based on gLite has been recently installed for testing purposes.

6 Summary

The generation and Geant4 simulation of events foreseen for ATLAS Data Challenges 2 have been completed using 3 flavors of Grid technology. They have been proven to be usable in a coherent way for a real production and this is a major achievement.

On the other hand, this exercise has taught us that all the involved elements (Grid middleware; production system; deployment and monitoring tools over the sites) need improvements.

Between the start of DC2 in July 2004 and the end of September 2004, the automatic production system has submitted about 235000 jobs. These jobs were approximately evenly distributed over the 3 Grid flavors. Overall, they consumed ~1.5 million SI2K months of cpu (~5000 cpu months on average present day cpu) and produced more than 30TB of physics data.

ATLAS is also pursuing a model for distributed analysis which would improve the productivity of end users by profiting from Grid available resources. Generic software components have been identified and several implementations of useful tools are being developed. An important aim is to provide the users with simple interfaces (ROOT, Python) that facilitate their interaction with the Grid infrastructure.

Acknowledgements

The authors would like to thank the many people involved in ATLAS DC2.

References

1. http://www.cern.ch/atlas
2. http://www.cern.ch/lhc
3. R. Sturrock et al. "ATLAS Data Challenge 1", CERN-PH-EP-2004-028, CERN, Apr 2004
4. http://lcg.web.cern.ch/LCG/
5. "The Grid 2003 Project." http://www.ivdgl.org/grid2003/index.php
6. "NorduGrid." http://www.nordugrid.org
7. http://www-cdf.fnal.gov
8. http://www-do.fnal.gov
9. L. Goossens, "Production System in ATLAS DC2", CHEP04, Interlaken, contr. no. 501
10. http://heppc12.uta.edu/windmill/
11. D. Rebatto, "The LCG Executor for the ATLAS DC2", CHEP04, Interlaken, contr. no. 364
12. R. Gardner, "ATLAS DC Production on Grid3", CHEP04, Interlaken, contr. no. 503
13. X. Zhao, "Experience with Deployment and Operation of the ATLAS production System and the Grid3", CHEP04, Interlaken, contr. no. 185
14. J. Kennedy, "Legacy services within ATLAS DC2", CHEP 2004, Interlaken, contr. no. 234
15. http://physics.bu.edu/pacman/
16. M. Branco, "Don Quijote", CHEP04, Interlaken, contr. no. 142
17. http://atlas.web.cern.ch/atlas/groups/software/DOCUMENTATION/ATLSIM/atlsim.html
18. http://atlas.web.cern.ch/atlas/GROUPS/SOFTWARE/OO/architecture/General/index.htm
19. http://www.thep.lu.se/~torbjorn/Pythia.html
20. http://hepwww.rl.ac.uk/theory/seymour/herwig/
21. R. Brun & F. Rademakers, "ROOT, An Object Oriented Data Analysis Framework",Proceedings AIHENP'96, Lausanne, Sep. 1996, Nucl. Inst. & Meth.A389 (1997) 81-86.
22. D. Duellmann, "The LCG POOL Project – General Overview and Project Structure" 2003
23. http://geant4.web.cern.ch/geant4/
24. "The European DataGrid project." http://eu-datagrid.web.cern.ch/
25. "Globus Toolkit." http://www-unix.globus.org/toolkit/
26. "Data TransAtlantic Grid." http://datatag.web.cern.ch/datatag
27. A. Chervenak, "Giggle: A Framework for Constructing Replica Location Services", SC02
28. http://www.usatlas.bnl.gov/ADA/
29. http://www.usatlas.bnl.gov/ADA/dels/ajdl/
30. D. L. Adams, "DIAL: Distributed Interactive Analysis of Large Datasets", CHEP03, UCSD
31. http://www.python.org/
32. http://lcg.web.cern.ch/LCG/peb/arda/
33. http://public.eu-egee.org/
34. http://glite.web.cern.ch/glite/
35. http://www.cs.wisc.edu/condor/
36. http://www.platform.com/products/LSFfamily/

High Throughput Computing for Spatial Information Processing (HIT-SIP) System on Grid Platform

Yong Xue[1,2], Yanguang Wang[1], Jianqin Wang[1], Ying Luo[1], Yincui Hu[1], Shaobo Zhong[1], Jiakui Tang[1], Guoyin Cai[1], and Yanning Guan[1]

[1] Laboratory of Remote Sensing Information Sciences,
Institute of Remote Sensing Applications, Chinese Academy of Sciences,
P. O. Box 9718, Beijing 100101, China
[2] Department of Computing, London Metropolitan University,
166-220 Holloway Road, London N7 8DB, UK
y.xue@londonmet.ac.uk, yx9uk@yahoo.com

Abstract. For many remote sensing application projects, the quality of the research or the product is heavily dependent upon the quantity of computing cycles available. Middleware is software that connects two or more otherwise separate applications across the Internet or local area networks. In this paper, we present the High Throughput Computing Spatial Information Processing (HIT-SIP) System (Prototype), which is developed in Institute of Remote Sensing Applications, Chinese Academy of Sciences, China. Several middleware packages developed in the HIT-SIP system are demonstrated. Our experience shows that it is feasible that our grid computing testbed can be used to do remote sensing information analysis.

1 Introduction

Dozens of satellites constantly collecting data about our planetary system 24 hours a day and 365 days a year. For many remote sensing application projects, the quality of the research or the product is heavily dependent upon the quantity of computing cycles available. Scientists and engineers engaged in this sort of work need a computing environment that delivers large amounts of computational power over a long period of time. Large scale of satellite data needed to be processed and stored in almost real time. So far this processing in remote sensing confronts much difficulties in one single computer, or even impossibility. Computing grid that is integrated by series of middleware provides a way to solve this problem.

It is not uncommon to find problems that require weeks or months of computation to solve. Such an environment is called a High-Throughput Computing (HTC) environment (*http://www.cs.wisc.edu/condor/*). The key to HTC is to efficiently harness the use of all available resources. Years ago, the engineering and scientific community relied on a large, centralized mainframe or a supercomputer to do computational work. A large number of individuals and groups needed to pool their financial resources to afford such a machine. Users had to wait for their turn on the mainframe, and they had a limited amount of time allocated. While this environment was inconvenient for users, the utilization of the mainframe was high; it was busy nearly all the time.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 40 – 49, 2005.

As computers became smaller, faster, and cheaper, users moved away from centralized mainframes and purchased personal desktop workstations and PCs. An individual or small group could afford a computing resource that was available whenever they wanted it. The personal computer is slower than the large centralized machine, but it provides exclusive access. Now, instead of one giant computer for a large institution, there may be hundreds or thousands of personal computers. This is an environment of distributed ownership, where individuals throughout an organization own their own resources. The total computational power of the institution as a whole may rise dramatically as the result of such a change, but because of distributed ownership, individuals have not been able to capitalize on the institutional growth of computing power. And, while distributed ownership is more convenient for the users, the utilization of the computing power is lower. Many personal desktop machines sit idle for very long periods of time while their owners are busy doing other things (such as being away at lunch, in meetings, or at home sleeping).

Grid computing enables the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities (Foster and Kesselman 1998, Foster *et al.* 2001). Just as an Internet user views a unified instance of content via the Web, a grid user essentially sees a single, large virtual computer. At its core, grid computing is based on an open set of standards and protocols — e.g., Open Grid Services Architecture (OGSA) — that enable communication across heterogeneous, geographically dispersed environments. With grid computing, organizations can optimize computing and data resources, pool them for large capacity workloads, share them across networks and enable collaboration. In fact, grid can be seen as the latest and most complete evolution of more familiar developments — such as distributed computing, the Web, peer-to-peer computing and virtualization technologies.

There are several famous grid projects today. Network for Earthquake Engineering and Simulation labs (NEESgrid) *(www.neesgrid.org)* intended to integrate computing environment for 20 earthquake engineering laboratories. Access Grid (*ww.fp.mcs.anl.gov/fl/access_grid*) lunched in 1999 and mainly focused on lecture and meetings-among scientists at facilities around the world. European Data Grid sponsored by European union, mainly in data analysis in high-energy physics, environmental science and bioinformatics. The Grid projects focused on spatial information include Chinese Spatial Information Grid (SIG) (*http://www.863.gov.cn*), ESA's SpaceGrid (*http://www.spacegrid.org*), and USA Earth System Grid (*http://www.earthsystemgrid.org*).

ESA's SpaceGrid is an ESA funded initiative. The SpaceGRID project is run by an international consortium of industry and research centres led by Datamat (Italy). Other members include Alcatel Space (France), CS Systemes d'Information (France), Science Systems Plc (UK), QinetiQ (UK) and the Rutherford Appleton Laboratory of the UK Council for the Central Laboatory of the Research Councils. Two other aspects of this project are of particular importance for ESA: finding a way to ensure that the data processed by the SpaceGRID can be made available to public and

educational establishments, and ensuring that SpaceGRID activities are coordinated with other major international initiatives.

The SpaceGRID project aims to assess how GRID technology can serve requirements across a large variety of space disciplines, such as space science, Earth observation, space weather and spacecraft engineering, sketch the design of an ESA-wide GRID infrastructure, foster collaboration and enable shared efforts across space applications. It will analyse the highly complicated technical aspects of managing, accessing, exploiting and distributing large amounts of data, and set up test projects to see how well the GRID performs at carrying out specific tasks in Earth observation, space weather, space science and spacecraft engineering.

The Earth System Grid (ESG) is funded by the U.S. Department of Energy (DOE). ESG integrates supercomputers with large-scale data and analysis servers located at numerous national labs and research centers to create a powerful environment for next generation climate research. This portal is the primary point of entry into the ESG.

ESG Collaborators include Argonne National Laboratory, Lawrence Berkeley National Laboratory, Lawrence Livermore National Laboratory, National Center for Atmospheric Research, Oak Ridge National Laboratory and University of Southern California/Information Sciences Institute.

The Condor Project has performed research in distributed high-throughput computing for the past 18 years, and maintains the Condor High Throughput Computing resource and job management software originally designed to harness idle CPU cycles on heterogeneous pool of computers. In essence a workload management system for compute intensive jobs, it provides means for users to submit jobs to a local scheduler and manage the remote execution of these jobs on suitably selected resources in a pool. Condor differs from traditional batch scheduling systems in that it does not require the underlying resources to be dedicated: Condor will match jobs (*matchmaking*) to suited machines in a pool according to job requirements and community, resource owner and workload distribution policies and may vacate or migrate jobs when a machine is required. Boasting features such as check-pointing (state of a remote job is regularly saved on the client machine), file transfer and I/O redirection (i.e. remote system calls performed by the job can be captured and performed on the client machine, hence ensuring that there is no need for a shared file system), and fair share priority management (users are guaranteed a fair share of the resources according to pre-assigned priorities), Condor proves to be a very complete and sophisticated package. While providing functionality similar to that of any traditional batch queuing system, Condor's architecture allows it to succeed in areas where traditional scheduling systems fail. As a result, Condor can be used to combine seamlessly all the computational power in a community.

In this paper, we present the High Throughput Computing Spatial Information Processing (HIT-SIP) System (Prototype), which is developed in Institute of Remote Sensing Applications, Chinese Academy of Sciences, China. The system will be introduced in Section 2. Several middleware developed in the HIT-SIP system will be demonstrated in Section 3. Finally, the conclusion and further development will be addressed in Section 4.

2 High Throughput Computing Spatial Information Processing Prototype System

Remote sensing data processing is characterized by magnitude and long period of computing. The HIT-SIP system on Grid platform has been developed in Institute of Remote Sensing Application, Chinese Academy of Science. It is an advanced High-Throughput Computing system specialized for remote sensing data analysis using Condor. Heterogeneous computing nodes including two sets of Linux computers and WINNT 2000 professional computers and one set of WINNT XP computer provide stable computing power (Wang *et al.* 2003, 2004). The grid pool uses java universe to screen heterogeneous characters. The function structure of HIP-SIP system is shown in Figure 1.

There are 3 different ways to access the HIT-SIP system: Web service, local pool and wireless Web service. The functionalities of the HIT-SIP are:

- Remote sensing image processing and analysis: Thematic information extraction.
- Spatial analysis: Overlay, buffering, Spatial Statistics
- Remote sensing applications: Aerosol, Thermal Inertial and Heat Flux Exchange
- System operations: System status, Job management
- Visualization

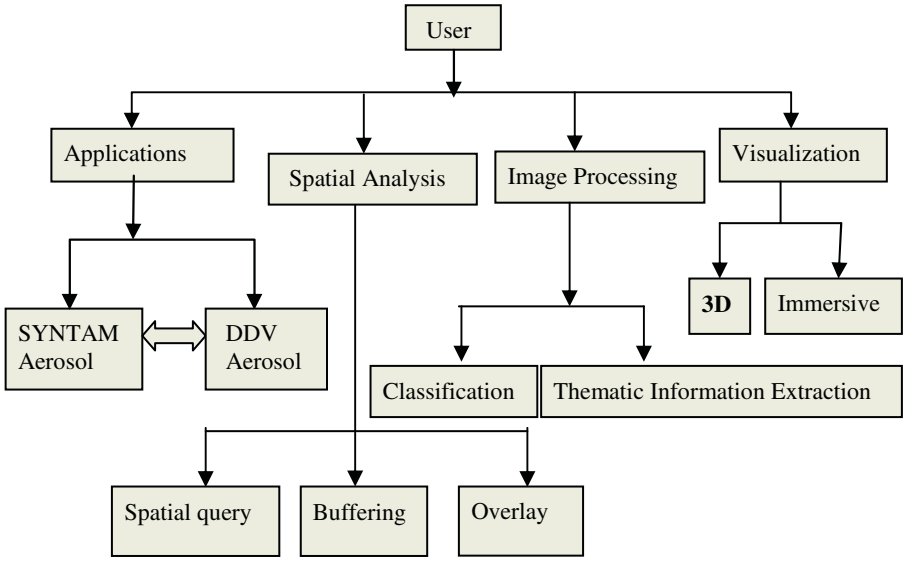


Fig. 1. The Functionality Structure of HIT-SIP system

The key technologies of HIT-SIP involve integrating Web Services with condor, namely a Grid implementation of Condor. There are about several building remote sensing applications on top of condor, as follows, command line tools of condor that are applied in JSP and Servlet, Web Services (Xue *et al*, 2002), and Condor GAHP

etc. In HIT-SIP, we serve those people who use web browsers on PCs or on mobile device, such as PDA, by using Java Servlet to invoke command line tools of condor in applications server Resin; and we provide programmer WSDL that tell how to invoking our grid computing functionality of Spatial Information Processing.

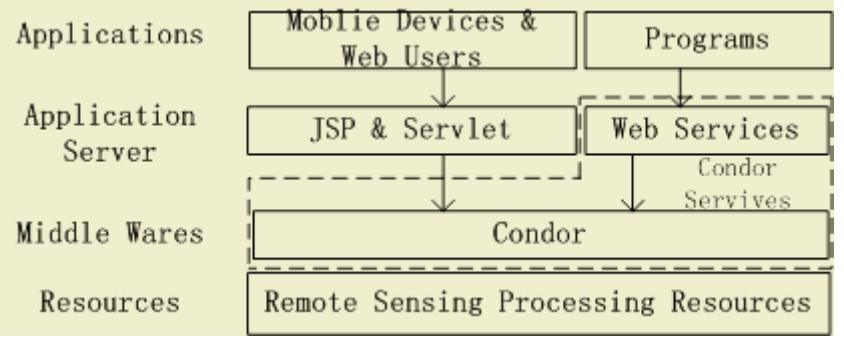


Fig. 2. The Structure of HIT-SIP system

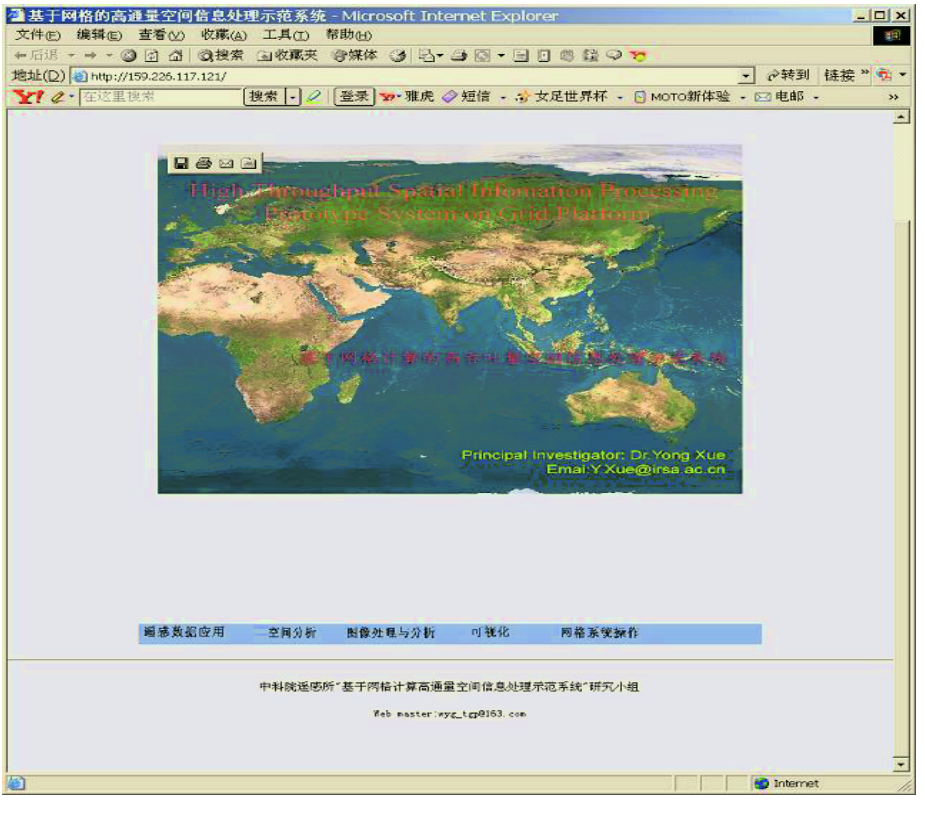


Fig. 3. The interface of front page of HIT-SIP system

Web Service is selected because compared to other distributed computing technologies, such as CORBA, RMI and EJB, it is a more suitable candidate for Internet scale application. First, web service is based on a collection of open standards, such as XML, SOAP, WSDL and UDDI. It is platform independent and programming language independent because it uses standard XML language. Second, web service uses HTTP as the communication protocol. That is a big advantage because most of the Internet's proxies and firewalls will not mess with HTTP traffic. Figure 2 shows the basic structure of HIT-SIP. Figure 3 show the interface of front page of HIT-SIP system, which can be access by web service.

3 Middleware Development for the HIT-SIP System

Middleware makes resource sharing seem transparent to the end user, providing consistency, security, privacy, and capabilities. Middleware is software that connects two or more otherwise separate applications across the Internet or local area networks (*www.nsf-middleware.org*). More specifically, the term refers to an evolving layer of services that resides between the network and more traditional applications for managing security, access and information exchange to

- Let scientists, engineers, and educators transparently use and share distributed resources, such as computers, data, networks, and instruments,
- Develop effective collaboration and communications tools such as Grid technologies, desktop video, and other advanced services to expedite research and education, and
- Develop a working architecture and approach that can be extended to the larger set of Internet and network users.

The HIT-SIP system is a fundamental bare computing platform and can not run remote sensing image processing program. So remote sensing data processing middleware running on grid pool is inevitable. Common users can use the heterogeneous grid and share its strong computing power to process remote sensing data with middleware as if on one supercomputer.

All the processing algorithms of the remote sensing images can also be divided into three basic categories in term of the area dealt with by the operation:

a) Point operators

Point operators change each pixel of the original image into a new pixel of the output image in terms of the given mathematics function, which transforms single sample point. The transform process doesn't refer to other samples and other samples don't take part in the calculation.

b) Local operators

Local operators transform single sample point and the transform process takes its neighbor samples to take part in calculation. The output value is related to itself and its neighbor pixels.

c) Global operators

Global operators incorporate the whole image's samples into the calculation when it copes with each sample.

There are three approaches to divide a sequential task into parallel subtasks: data partition, task partition and mix partition (Cai *et al.* 2004, Hu *et al.* 2004a, b). The data partition parts the image data into a number of sub-data. Each subtask processes one sub-data respectively. In this case, the function of each subtask is identical, and only the processing data is different. The task partition parts the process of computing into a number of steps. The efficiency can be increased through parallel execution of these steps. The mix partition take the above two approaches into account to design the parallel algorithm. The Grid computing takes the Internet as a computing platform, thus it differs from additional parallel computing or distributed computing. We must consider the bandwidth of the network, stability, global distribution of the nodes, security etc. the data partition is a suitable approach. If the algorithm cannot be designed by means of the data partition. We can try to part it into relatively unattached (little interaction) subtasks. If it cannot be done, we will conclude that Grid computing is not fit for the algorithm according to the above given criteria. Through analyzing all sorts of algorithms, we conclude that those algorithms which pertain to point operators or local operators are easy to part it them into relative unattached subtasks, however, for global operators, only if we take special data partition or together with task partition, We can adapt it to meet the condition of using grid computing.

3.1 Middleware for Aerosol Retrieval from MODIS Data

GCP-ARS (Grid Computation Platform for Aerosol Remote Sensing), developed by Telegeoprocessing Research group in Institute of Remote Sensing Applications (IRSA), Chinese Academy of Sciences (CAS), is a advanced high throughput computing grid middleware that supports the execution of aerosol remote sensing retrieval over a geographically distributed, heterogeneous collection of resources (Tang *et al.* 2004a, b). In GCP-ARS, Condor system provides the technologies for resource discovery, dynamic task-resource matching, communication, and result collection, etc.

LUTGEN module provides the users of the power to generate the LUT on grid computing platform, LOOK-UP module can be used to look up and interpolate the LUT generated by LUTGEN module or user self-offered to determine aerosol optical thickness. GCP-ARS provides the users friendly Windows-based GUI (Graphical User Interface), computation tasks auto-partitioned, tasks auto-submission, the execution progress monitoring, the sub-results collection and the final result piecing, which results in screening the complexities of grid applications programming and the Grid platform for users. GCP-ARS architecture mainly consists of three entities: clients, a resource broke, and producers. Aerosol retrieval is launched though a client GUI for execution at producers that share its idle cycles through the resource broker. The resource broke manages tasks application execution, resource management and result collection by means of Condor system.

The date and time of MODIS data we have chose for our test on GCP-ARS was acquired at 03:20 UTC on August 11, 2003. The image size is 512 x 512 with spatial resolution 500*m*, which covers most part of plain of north China and Bohai gulf. Figure 4 shows the final AOT (Aerosol Optical Thickness) image retrieved using GCP-ARS.

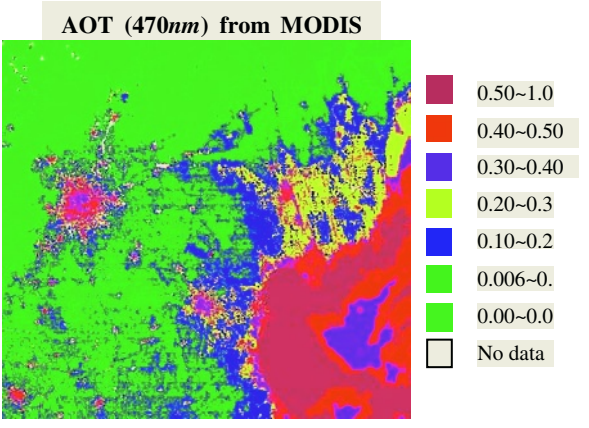


Fig. 4. AOT Image of MODIS Band 3 at 470 *nm* retrieved using GCP-ARS

3.2 Middleware for NDVI Calculation from MODIS `Data

Vegetation indices (VIs) are spectral transformations of two or more bands designed to enhance the contribution of vegetation properties and allow reliable spatial and temporal inter-comparisons of terrestrial photosynthetic activity and canopy structural variations. The Normalized Difference Vegetation Index (NDVI), which is related to the proportion of photosynthetically absorbed radiation, is calculated from atmospherically corrected reflectances from the visible and near infrared remote sensing sensor channels as: (CH2 - CH1) / (CH2 + CH1), where the reflectance values are the surface bidirectional reflectance factors for MODIS bands 1 (620 - 670 nm) (CH1) and 2 (841 - 876 nm) (CH2). Figure 5 shows the architecture of the Condor pool supported mobile geoprocessing system. Figure 6 is one of the GUIs.

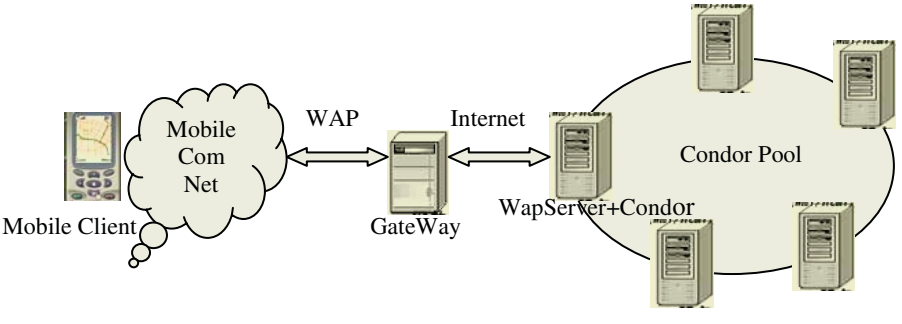


Fig. 5. The architecture of the Condor pool supported mobile geoprocessing system

One track of MODIS data (800x5000) used for the calculation of NDVI was acquired on 11 August 2003. We submitted the job from a WAP mobile phone to our

Condor pool. Figures 6 show the interface on WAP phone. We divided whole images into 4 parts. In theory, we could divide them into any number of parts. The 4 jobs have been submitted to our Condor pool. After each calculation, the results will be merged into one image (Wang *et al*. 2004). All these operations were performed automatically.

Fig. 6. The interface on WAP phone

4 Conclusions

Grid computing and Web Service technology is really a very effective method for sharing remote sensing data and processing middleware codes. It's feasible that our grid computing testbed can be used to do remote sensing information modelling. The most efficient image size and the best separation of a large image that apt to our testbed will be our further research with realizing an algorithm that can read and divide remote sensing images automatically, and transfer results back to the submitted machine as a whole data file.

Acknowledgement

This publication is an output from the research projects "CAS Hundred Talents Program" funded by Chinese Academy of Sciences, "Grid platform based aerosol fast monitoring modeling using MODIS data and middlewares development" (40471091) funded by National Natural Science Foundation of China (NSFC), and "Dynamic Monitoring of Beijing Olympic Environment Using Remote Sensing" (2002BA904B07-2) and "863 Plan – Remote Sensing Data Processing and Service Node" funded by the Ministry of Science and Technology, China.

Reference

[1] I. Foster, "What is the Grid: A Three-Point Checklist", *Grid Today*, 1 (6), 2002.
[2] I. Foster, C. Kesselman (Eds), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1998.

[3] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of Super-computer Applications*, 15(3), 2001, pp. 200-222.

[4] Jianqin Wang, Yong Xue, and Huadong Guo, 2003, A Spatial Information Grid Supported Prototype Telegeoprocessing System. In Proceedings of 2003 IEEE International Geoscience and Remote Sensing Symposium (IGARSS'2003) held in Toulouse, France on 21-25 July 2003, v 1, p 345-347.

[5] Yincui Hu, Yong Xue, Jianqin Wang, Guoyin Cai, Shaobo Zhong, Yanguang Wang, Ying Luo, Jiakui Tang, and Xiaosong Sun, 2004, Feasibility Study of Geo-Spatial Analysis using Grid Computing. *Lecture Notes in Computer Science,* Vol. 3039, pp.971-978.

[6] Guoyin Cai, Yong Xue, Jiakui Tang, Jianqin Wang, Yanguang Wang, Ying Luo, Yincui HU, Shaobo Zhong, and Xiaosong Sun, 2004, Experience of Remote Sensing Information Modelling with Grid computing. *Lecture Notes in Computer Science,* Vol. 3039, pp.1003-1010.

[7] Jianqin Wang, Xiaosong Sun, Yong Xue, Yanguang Wang, Ying Luo, Guoyin Cai, Shaobo Zhong, and Jiakui Tang, 2004, Preliminary Study on Unsupervised Classification of Remotely Sensed Images on the Grid. *Lecture Notes in Computer Science,* Vol. 3039, pp.995-1002.

[8] Yanguang Wang, Yong Xue, Jianqin Wang, Ying Luo, Yincui HU, Guoyin Cai, Shaobo Zhong, Jiakui Tang, and Xiaosong Sun, 2004, Ubiquitous Geocomputation using Grid technology – a Prime. In *Proceedings of IEEE/IGARSS to be held at Anchorage, Alaska, 20-24 September 2004*.

[9] Yincui Hu, Yong Xue, Jianqin Wang, Guoyin Cai, Yanguang Wang, Ying Luo, Jiakui Tang, Shaobo Zhong, Xiaosong Sun, and Aijun Zhang, 2004, An experience on buffer analyzing in Grid computing environments. In *Proceedings of IEEE/IGARSS to be held at Anchorage, Alaska, 20-24 September 2004*.

[10] Jiakui Tang, Yong Xue, Yanning Guan, Linxiang Liang, Yincui Hu, Ying Luo,Guoyin Cai, Jianqin Wang, Shaobo Zhong, Yanguang Wang, Aijun Zhang and Xiaosong Sun, 2004, Grid-based Aerosol Retrieval over Land Using Dark Target Algorithm from MODIS Data. In *Proceedings of IEEE/IGARSS to be held at Anchorage, Alaska, 20-24 September 2004*.

[11] Jiakui Tang, Yong Xue, Yanning Guan, and Linxiang Liang, 2004, A New Approach to Generate the Look-Up Table Using Grid Computing Platform for Aerosol Remote Sensing. In *Proceedings of IEEE/IGARSS to be held at Anchorage, Alaska, 20-24 September 2004*.

[12] Gang Xue, Matt J Fairman, Simon J Cox, 2002, Exploiting Web Technologies for Grid Architecture. In Proceedings of 2002 Cluster Computing and the Grid, pp.272-273.

The University of Virginia Campus Grid: Integrating Grid Technologies with the Campus Information Infrastructure*

Marty Humphrey and Glenn Wasson

Department of Computer Science, University of Virginia,
Charlottesville, VA 22904
{humphrey, gsw2c}@cs.virginia.edu

Abstract. Grid software often unfortunately requires significant changes in existing infrastructure, both in terms of policy and mechanism, instead of accommodating and leveraging existing information servers such as enterprise LDAP servers and enterprise authentication infrastructures. The University of Virginia Campus Grid (UVaCG) has been designed explicitly to re-use as much existing infrastructure in the campus environment as possible in creating a Grid based on the Web Services Resource Framework (WSRF), specifically the Globus Toolkit v4 and WSRF.NET. We report on the design and the current status of the UVaCG, with particular emphasis on the challenge of creating explicit policy expression, negotiation, and enforcement. When fully operational, campus researchers will be able to seamlessly utilize resources within the campus enterprise and expand on-demand to larger Grids such as the TeraGrid and the Open Science Grid.

1 Introduction

In many ways, the core challenge of Grid computing is the ability to seamlessly integrate legacy systems, both in terms of policy and mechanism. That is, while new protocols are often needed for such Grid-specific activities as remote execution and third-party data transfer, the goal of the Grid is to create a virtual computing platform from *existing* heterogeneous systems without requiring the imposition of new software mechanisms and policies that are radically different than those already in place. For example, it is unrealistic to expect sites to abandon an existing Kerberos authentication infrastructure [1] in lieu of some Grid-specific authentication such as the PKI-based GSI [2]. Similarly, resource owners who wish to participate in Grid activities must absolutely be allowed to retain control over their resources in a manner in which

* This work is supported in part by the US National Science Foundation under grants ACI-0203960 (Next Generation Software program), SCI-0438263 (NSF Middleware Initiative), SCI-0123937 (through a subcontract to SURA), the US Department of Energy through an Early Career Grant (Humphrey), and Microsoft Research.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 50 – 58, 2005.

they have previously established. Without this support for local autonomy, the vision of the Grid as vital-yet-commonplace infrastructure will never be achieved.

The "campus Grid" is a compelling environment for Grid activities, because of the inherent opportunity for sharing of resources between campus researchers. Campus researchers should be able to utilize unused campus-wide resources such as the PCs available in general-purpose labs. However, mechanisms are lacking by which to integrate campus IT infrastructure and policies—both explicit and implicit—with the current Grid tools, particularly those based on the Web Services Resource Framework (WSRF [3][4]). Maintaining student security and privacy (as defined by FERPA [5] and increasingly HIPAA [6]) is paramount. While Grid software is certainly being deployed on machines on campuses as part of national and international Grid efforts, these deployments are generally "side-by-side" with the main campus IT infrastructure, particularly that for AAA (authentication, authorization, and accounting). In short, the campus is an excellent candidate for Grids, but only if the campus is committed to it, the Grid software taps into existing campus IT infrastructures, and the Grid deployment is especially careful not to impede pre-existing use patterns by campus researchers and students.

This paper describes the design and current status of the University of Virginia Campus Grid (UVaCG), whose goal is to both create an intra-campus environment of sharing and as such facilitate the broader inter-campus sharing via future participation in Grids such as the Open Science Grid [7] and the TeraGrid [8]. This will be the first Grid in which the Windows machine (via our own WSRF.NET) participates side-by-side with the Linux machine (via the Globus Toolkit v.4). The key challenges will be for the Grid infrastructure to recognize and leverage the pre-existing campus trust fabric and campus policies. We describe the technologies being utilized, the unique challenges being faced as we attempt to integrate a substantial campus IT with the Grid technologies particularly with regard to policy management, and present a status report of the UVaCG. We believe that the UVaCG will be a prototypical example of a new class of Grids -- the enterprise Grid that is relatively self-contained but which interoperates and extends into other Grids on demand. As described in this paper, the key will be expressible, manageable, and negotiable policies on the part of resource owners, service deployers, service invokers, and the virtual organizations themselves.

This paper is organized as follows. In Section 2, we present the technologies being used and deployed for the UVaCG. Section 3 enumerates the challenges being encountered and addressed. Section 4 contains the conclusions.

2 Core Technologies

In this section, we describe the core technologies that are the basis of the University of Virginia Campus Grid (UVaCG). Section 2.1 discusses the Grid-related technologies, Section 2.2 discusses the key components of the Campus IT, and Section 2.3 illustrates how everything fits together.

2.1 Grid Technologies

The foundation of the University of Virginia Campus Grid (UVaCG) is the Web Services Resource Framework (WSRF). WSRF is a set of specifications that describe the relationship between "stateful resources" and web services. This relationship is defined in terms of WS-Resources, an abstraction for modeling/discovering state manipulated by web services. A WS-Resource is a "composition of a web service and a stateful resource" [3] described by an XML document (with known schema) that is associated with the web service's port type and addressed by a WS-Addressing EndpointReference [9]. WSRF defines functions that allow interactions with WS-Resources such as querying, lifetime management and grouping. WSRF is based on the OGSI specification [10] and can be thought of a expressing the OGSI concepts in terms that are compatible with today's web service standards [11]. Arguably and simplistically, it is sometimes convenient when contrasting OGSI and WSRF to think of OGSI as "distributed objects that conform to many Web Services concepts (XML, SOAP, a modified version of WSDL)", while WSRF is fundamentally "vanilla" Web Services with more explicit handling of state. One artifact of this is that OGSI did not really support interacting with these base Web Services and instead only interacted with "Grid Services" (by definition these were OGSI-compliant); WSRF fully supports interacting with these base Web Services (although the argument is made that client interactions with WSRF-compliant are richer and easier to manage).

Currently, there are 4 specifications [4] in the WS-ResourceFramework with a small number yet to be officially released. WS-ResourceProperties defines how WS-Resources are described by ResourceProperty (XML) documents that can be queried and modified. WS-ResourceLifetime defines mechanisms for destroying WS-Resources (there is no defined creation mechanism). WS-ServiceGroups describe how collections of services can be represented and managed. WS-BaseFaults defines a standard exception reporting format. WS-RenewableReference (unreleased) will define how a WS-Resource's EndpointReference, which has become invalid, can be refreshed. There are also 3 WS-Notification specifications (WS-BaseNotification, WS-Topics and WS-BrokeredNotification) that although not part of WSRF, build on it to describe asynchronous notification.

UVaCG will rely on two packages that support WSRF and WS-Notification: the Globus Toolkit v4 [12] and our own WSRF.NET [13][14]. We have been working closely with the Globus Toolkit developers to ensure that the two systems are interoperable, both with regard to the core WSRF and WS-Notification specifications and with regard to the Globus protocols.

Many of the existing Windows boxes on campus that are to be included in the UVaCG are already controlled by a Windows domain (username/password) that is maintained by the UVa Campus IT department. To support single sign-on and re-use existing authentication infrastructures, we have developed CredEx [15], which is a general-purpose credential exchange mechanism. The main authentication mechanism will be the campus PKI (described in the next section), and these PKI credentials will be dynamically exchanged to obtain the appropriate username and password for the target (Windows) machine. We would have liked to use MyProxy [16], but MyProxy

is not capable of speaking SOAP and WS-Security. By using CredEx, campus researchers can still rely on the Campus IT department to create and maintain accounts; in the UVaCG we are selectively and securely re-using this legacy infrastructure.

2.2 Campus Technologies

In addition to leading a well-run campus infrastructure consisting of network management, user-help desk, account creation, etc., the University of Virginia is very active in the Internet2 community, helping to develop and promote common schemas (such as EduPerson and LDAP Recipe) and PKI procedures for inter-campus interoperability.

Our main focus with the UVaCG is the re-use of campus authentication techniques. Recently, we have pursued the Bridge Certificate Authority (CA) as the basis for scalable Grid PKIs [18]. The Bridge CA is a compromise between a strictly hierarchical PKI and a mesh PKI and achieves many of the benefits of the hierarchical PKI and mesh PKI without with single point of failure of the hierarchical PKI and without the path validation complexity of the mesh PKI. This work is in part based on our previous work contributing to the Internet2/EDUCAUSE HEPKI-TAG group [19] and the EDUCAUSE effort to build and deploy a Higher Education Bridge CA (HEBCA) [20] in the US. Because the University of Virginia runs the PubCookie [21] infrastructure, in which a cookie-based authentication can be used to protect content on web servers (e.g., as an alternative for password-protected course content), we are in the final stages of completing a prototype in which a valid PubCookie can be exchanged (via CredEx discussed in the previous section) for either a username/password for a target machine or for a valid X.509 proxy credential for the Grid.

Our second major focus is the re-use of campus account creation and management. We are currently considering the use of Walden [17] to manage the campus-wide Gridmap file as well as allocate temporary identities and accounts for non-local researchers.

2.3 University of Virginia Campus Grid (UVaCG)

By re-using campus authentication and campus account procedures, the WSRF implementations of the Globus Toolkit and WSRF.NET will be deployed with the minimal amount of disruption for existing campus resource users. The campus Grid resources—Windows machines *and* Linux compute clusters—will be seamlessly available to University of Virginia researchers and their collaborators as appropriate. Our goal is to have campus researchers that acquire new machines want to have these resources included in the campus grid, because they know that they are *not*, in fact, "giving their cycles away" but are instead being given access to an instantaneous pool of resources that they could never acquire merely in their campus lab. Figure 1 shows the UVaCG, illustrating the resources involved and the Grid users. UVaCG initially, will contain five separately-owned Linux clusters and two separately-owned instructional labs containing Windows XP machines.

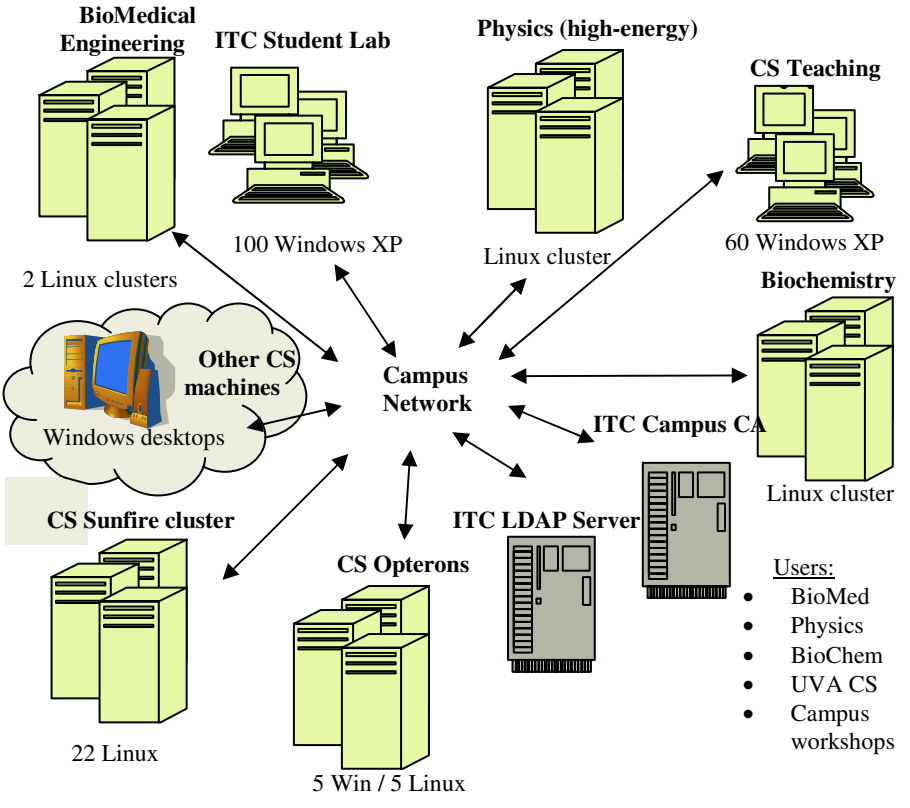


Fig. 1. University of Virginia Campus Grid (UVaCG)

3 Explicit Policy Expression, Negotiation, and Enforcement for the Campus Grid

As we make progress on the interoperability between WSRF.NET and the Globus Toolkit v4, and we successfully integrate with the campus IT environment, we are finding it necessary to be able to express policies on these services, both from the viewpoint of the service author and/or deployer, the administrator of the domain in which the service is deployed, as well as the hierarchical policy makers of the Grid (Grid users will also have policies). These policies must be expressed and obeyed. We believe that in today's Grid deployments, policies of Grid resource owners and Grid users are unnecessarily simplistic and homogeneous. For example, machines will often be purchased as part of a funded project and dedicated to "The Grid". Resources will be hand-selected or hard-coded by the Grid user because he/she has had a good experience with that particular resource in the past. We believe that individual Grid users can have individual policies, and they want the Grid automation tools to adhere to their policies. Grid resource owners also want to explicitly and concisely express their policies, *if even only to make it possible for Grid users (or tools acting on their*

behalf) to begin to recognize these policies and adhere to them. Today, this is not possible because policy languages have not been evaluated, policy negotiation tools don't exist, and policy enforcement points are ad hoc, if they exist at all.

We have begun investigating approaches by which Grid users, resource owners, and the Grid as a whole can begin to express these policies and have them adhered to and enforced [22]. We focused only on the *resource provisioning policy,* which describes the Grid-wide distribution of resource utilization. In our approach, we made policy operational by describing actions that will be taken to maintain the desired Grid state. For example, a policy such as "if any site is performing less than 25% of the work of the other sites, all new work will be scheduled on that site until the work load is equalized" describes an explicit *trigger condition* and an *action* that will be taken if that condition is met. Note that while such statements may allow for automated policy enforcement, the Grid's response need not be "fully automated". That is, the appropriate administrator could be alerted to the relevant condition and/or a set of corrective actions might be suggested, allowing the human to make the final decision.

We have focused our preliminary explorations for the UVaCG on three representative Grid-wide resource provisioning policies:

- Each physical organization opportunistically gives what it can to the Grid [the you-give-what-you-can policy]
- Resource utilization is divided equally among member resources [the 1/N policy]
- Each physical organization receives Grid utilization credit for the resource utilization their physical organization provides to other Grid users outside their physical organization [the you-get-what-you-give policy]

Our prototypical software simulation implemented the "you-get-what-you-give" policy and consisted of three types of software components: *Gatekeeper services, Bank services*, and *Enforcement services*. The Gatekeeper service was based on the operations of the Globus gatekeeper and controlled access to the resources. The bank service recorded the "contribution" the physical organizations made to the Grid (based on a hypothetical exchange rate). The enforcement services implemented two kinds of enforcement actions in the Grid, one punitive ("cutoff") and one corrective ("redirect"). The "cutoff" action is when the enforcer contacts one or more Gatekeepers and instructs them to deny access to their resources to the enforcer's associated user. The "redirect" action involves the enforcer contacting one or more Gatekeepers to ask that resource requests they receive be re-directed to the enforcer's user's resource. This has the effect of providing more credit to a needy user.

While arguably simplistic, we were able to conduct experiments in which we refined the issues and approach and began to quantify the costs and benefits of attempting to enforce Grid policy in this manner. In our experiments, we were able to dynamically shift the workload to meet the resource-sharing policy, but only under highly-constrained conditions. *We believe that policy expression, negotiation, and enforcement is an extremely difficult problem that must be addressed comprehensively in Grids.* Specifically, we are extending this work in the following ways for the UVaCG:

- **Expressible policies.** We will make representative policies for resource owners and Grid users using a combination of WS-Policy, WS-SecurityPolicy, and XACML. Our previous policies were in our own ad-hoc language.
- **Policy enforcement in WSRF.NET**. While we plan to utilize the Microsoft WSE as much as possible for policy enforcement, and we suspect that much of the policy enforcement may only be performed via application-specific logic, we believe that the WSRF.NET hosting environment that we are constructing can perform policy enforcement for the entire PC. An important first application of this is to enforce the resource owner's policy (such as "If a local user is interacting with this PC, no Grid activities can execute on this PC.") We will create comprehensive, integrated logging for post-facto analysis.
- **Prototype tools that understand explicit policies.** The first tool that we will develop will be a scheduler that understands explicit policies. A core capability of this tool will be the hypothetical exploration of the Grid via "What if…?" questions, such as "What if I tried to execute my parameter-space job at 5pm? Where could I be allowed to execute?" A related tool will be used to understand the effects of policy on the Grid. For example, we anticipate situations in which policies are so restrictive (e.g., by a resource owner) that *no* services are engaged or *no* jobs are executed. Breaking the "inertia" of first Grid deployment, as well as determining the steady-state performance/behavior of the Grid will be very challenging problems. We need tools to resolve these policy conflicts and make suggested modifications.

We are leveraging existing work to express, negotiate, and enforce the policies of Campus Grid users and resource providers. To a certain extent, security authorization policy has been treated explicitly in Grid contexts in Akenti [23], CAS [24] and the work by Keahey and Welch [25]. These systems make permit/deny decisions based on (potentially multiple) access control policies and the accessor's identity / group membership. While this project will leverage these excellent projects, this existing work does not immediately satisfy the requirements of the UVaCG because all focus on authorization policy, whereas we need to address policies more broadly and in the context of Web Services (e.g., "I prefer predictable Grid behavior rather than having 9 out of 10 jobs finish very quickly and the last job finish incredibly slowly"). Also, the referenced work is generally applicable to a *particular service*, whereas the problem we are addressing encompasses *per-Grid-user, per-Service,* and *Grid-wide.* For example, the campus IT administrators may impose requirements on the "holistic" behavior of the Grid that is not merely satisfied through the individual actions of the services and the Grid users.

The UVaCG leverages research into explicit policy management that has been performed outside of the context of the Grid. Preconditions and obligation in policy [26] provide a compact means of representing what a user must do *before* performing a specific action as well as what they must do *after* an action. The Generic Authentication, Authorization, and Accounting (AAA) Architecture (RFC 2903[27]) builds an architecture in terms of Generic AAA servers, policy and event repositories, and Application Specific Modules (ASMs). It is not clear how this architecture can be applied to policies broader than AAA. The specification is also not prescriptive in how to best *implement* such an architecture based on a particular technology such as Web Services.

Within the context of OGSA, the WS-Agreement specification proposes support for service management, which is "the ability to create Grid services and adjust their policies and behaviors based on organizational goals and application requirements"[28]. This is a form of Service-Level Agreement (SLA) for OGSA. WS-Agreement will be leveraged as much as possible to frame the work described in this proposal. While WS-Agreement does not suggest actual policies for Grid users, Grid Services, and the Grid as a whole (which is a contribution of this work), we anticipate using aspects of WS-Agreement.

4 Conclusion

The campus Grid presents a unique set of challenges and opportunities with regard to Grid Computing. Explicit policy management and negotiation is needed in order to support the dynamic environment. We are currently continuing to advance the interoperability of WSRF.NET and the Globus Toolkit v4 and have begun initial deployment on the UVaCG.

We are also separately working with the San Diego Supercomputing Center (SDSC) to develop support for the campus researcher to more easily handle the stringent security requirements of SDSC, NCSA, and the TeraGrid. We are hardening our Bridge CA work, whereby campuses that meet the security requirements of the TeraGrid can "cross-certify" with other CAs that the TeraGrid recognizes. This will, in principle, allow campuses to utilize their Campus authentication infrastructure as the basis of a single sign-on capability.

References

[1] B.C. Neuman, and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33-38, September 1994.

[2] Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pg. 83-92, 1998.

[3] K. Czajkowski., Ferguson, D., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W. 2004. The WS-Resource Framework. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf

[4] WS-ResourceFramework and WS-Notification Specifications. http://devresource.hp.com/drc/specifications/wsrf/index.jsp

[5] Family Educational Rights and Privacy Act (FERPA). US Department of Education. http://www.ed.gov/policy/gen/guid/fpco/ferpa/index.html

[6] United States Department of Health and Human Services. Office of Civil Rights – HIPAA. http://www.hhs.gov/ocr/hipaa/

[7] Open Science Grid. http://www.opensciencegrid.org/

[8] TeraGrid. http://www.teragrid.org

[9] IBM, BEA, and Microsoft. WS-Addressing. 2004. http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-addressing.asp

[10] S. Tuecke et. al. Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum. GFD-R-P.15. Version as of June 27, 2003.
[11] K. Czajkowski, Ferguson, D., Foster, I., Frey, J., Graham, S., Snelling, D., Tuecke, S., From Open Grid Services Infrastructure to Web Services Resource Framework: Refactoring and Evolution, 2004. http://www-106.ibm.com/developerworks/webservices/library/ws-resource/grogsitowsrf.html
[12] Globus Toolkit v. 4. http://www.globus.org/wsrf/
[13] WSRF.NET: The Web Services Resource Framework on the .NET Framework. http://www.ws-rf.net
[14] Humphrey, M., G. Wasson, M. Morgan, and N. Beekwilder (2004). An Early Evaluation of WSRF and WS-Notification via WSRF.NET. *2004 Grid Computing Workshop (associated with Supercomputing 2004).* Nov 8 2004, Pittsburgh, PA.
[15] D. Del Vecchio, J. Basney, N. Nagaratnam, and M. Humphrey. CredEx: User-Centric Credential Selection and Management for Grids. University of Virginia Computer Science Technical Report. November, 2004.
[16] J. Novotny, S. Tuecke, and V. Welch. An Online Credential Repository for the Grid: MyProxy. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
[17] Kirschner, B., Adamson, W., Hacker, T. , Athey, B., Walden: A Scalable Solution for Grid Account Management, *2004 Grid Computing Workshop (associated with Supercomputing 2004).* Nov 8 2004, Pittsburgh, PA.
[18] J. Jokl, J. Basney, and M. Humphrey. Experiences using Bridge CAs for Grids. UK Workshop on Grid Security Experiences, Oxford 8th and 9th July 2004.
[19] Higher Education PKI Technical Activities Group (HEPKI-TAG). http://middleware.internet2.edu/hepki-tag/
[20] Higher Education Bridge Certificate Authority (HEBCA). http://www.educause.edu/hebca/
[21] Pubcookie. http://www.pubcookie.org
[22] G. Wasson and M. Humphrey. Policy and Enforcement in Virtual Organizations. In *4th International Workshop on Grid Computing (Grid2003)* (associated with Supercomputing 2003). Phoenix, AZ. Nov 17, 2003.
[23] M. Thompson. 2001. Akenti Policy Language. http://www-itg.lbl.gov/security/Akenti/Papers/PolicyLanguage.html
[24] L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke. A Community Authorization Service for Group Collaboration. *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.
[25] K. Keahey, V. Welch. Fine-Grain Authorization for Resource Management in the Grid Environment. *Proceedings of Grid2002 Workshop*, 2002.
[26] N. Dulay, Lupu, E., Sloman, M. and Damianou, N. 2001. A Policy Deployment Model for the Ponder Language. *Proc. IEEE/IFIP International Symposium on Integrated Network Management (IM'2001)*
[27] C. de Laat et. al. Generic AAA Architecture. RFC 2903. Available at: http://www.faqs.org/rfcs/rfc2903.html
[28] K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, and M. Xu. Agreement-based Service Management (WS-Agreement). Global Grid Forum draft-ggf-graap-agreement-1. Version as of Feb 8, 2004.

M-Grid: Using Ubiquitous Web Technologies to Create a Computational Grid

Robert John Walters and Stephen Crouch

Declarative Systems and Software Engineering Group,
Department of Electronics and Computer Science,
University of Southampton, UK
SO17 1BJ
{rjw1, stc}@ecs.soton.ac.uk

Abstract. There are many potential users and uses for grid computing. However, the concept of sharing computing resources excites security concerns and, whilst being powerful and flexible, at least for novices, existing systems are complex to install and use. Together these represent a significant barrier to potential users who are interested to see what grid computing can do. This paper describes M-grid, a system for building a computational grid which can accept tasks from any user with access to a web browser and distribute them to almost any machine with access to the internet and manages to do this without the installation of additional software or interfering with existing security arrangements.

1 Introduction

Probably the most widespread application of grid technology is a computational grid [2, 13, 14] which provides a network for the distribution of computational power available from connected machines to users with work to perform. On identifying a task, a user uses local software to present their task to the grid system which then arranges for the task to be executed (usually elsewhere and often distributed amongst a number of machines) and delivers the results back to the user. Existing systems require the installation of software onto the machines which are connected to form the grid.

The grid software [2, 7, 9-11, 13-15] provides the necessary infrastructure for the users to inject their tasks into the system, for the system to distribute tasks to nodes within the grid and to return the results. This software also implements mechanisms to transmit tasks, input data and results from machine to machine as necessary and to execute tasks when they arrive at remote machines. The effort associated with installing and configuring most of these systems is considerable. Aside from the effort associated with installing, configuring and coordinating the software, this can be a problem for two reasons. The first is that, depending on their environment, users may need to obtain permissions to install software onto the machines to be used. Even where this is not a problem, there is a second issue of the risk for those machines which will perform the computations of executing an unknown program which is delivered by the grid software but whose ultimate source is a third party. Handling these two issues adds considerable complexity to the task of installing a computa-

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 59 – 67, 2005.

tional grid, even where a lightweight grid framework [17] is being used. M-grid offers a solution which avoids these issues by using software and associated security which are ubiquitous.

The essence of a computational grid is that it provides a mechanism whereby a collection of computers with processing capability is made available to users with a computational tasks to perform. Each system has its own features and strengths, but they all have the following three roles performed by one or more of the machines which form the grid:

- Nodes which provide processing power to the grid.
- Users who supply tasks to be performed.
- Coordinator(s) who distribute incoming tasks to the processing nodes and see that results are returned as appropriate.

In a classical computational grid, there will be many nodes providing processing power and relatively few users each of whom has a need for large amounts of processing power to perform either a few large tasks (which are often broken into pieces for execution) or a great many relatively smaller ones. The coordination role is often performed by a single machine. The coordinator is at the heart of the system and has two main activities: collecting tasks and distributing them to available nodes. Discharging these obligations implies a subsidiary task of collecting the results and delivering them to the users.

The coordinating machine(s) need to be able to communicate with others in the grid and, in the case of the nodes providing processing, supply code (and associated data) for execution. This is typically achieved by installing grid software on each of the machines taking part which handles communications between the component parts of the grid. Installing this software can be problematic because of what it does. Owners of computers are concerned to ensure that new software installed will not disrupt, compromise or damage their system. Hence, before they are prepared to install it, they subject it to scrutiny to check that it's behaviour is acceptable. Areas of concern include ensuring that the software won't damage the machine on which it is installed, won't interfere with existing applications which operate on the machine and doesn't seek to destroy data or collect and disclose confidential information. This might be established directly by examination of the software, or indirectly by the certification by some trusted party. The aspect of the grid whereby the coordinator sends code to other nodes for execution is particularly problematic since the nature and detail of the code to be executed is not known at the time the grid infrastructure is being installed. Thus, in accepting code to execute, a node has to rely on the coordinator to only send code which is acceptable. This requires trusting relationships between the node, the coordinator and the client [12]. Establishing these trusting relationships can be both difficult and time consuming.

2 Why Is M-Grid So Easy?

In all of the above, the real nub of the problem is the danger associated with executing code which arrives via the network. These tasks arrive via the coordinator from clients who may not even be known to the processing node. In accepting this work, the

processing node has to trust that it can execute the code safely. In order to do this, the node must be prepared to trust in the other elements of the system. In the case of the client, the node has to either trust it directly or be prepared to rely on the coordinator to vet clients and the tasks they supply appropriately. In addition, the node then has to be prepared to trust the coordinator not to interfere with the tasks (such as by adding in its own unacceptable code).

We identified that, although the motivation is quite different, a browser which downloads and runs an applet contained within a web page is doing exactly what causes the trouble in a computational grid - the server sends code which is executed on the client machine in the browser [8, 16]. The enabling technology for this activity was developed to further the capability of the world wide web to provide a rich and comprehensive experience to the user of the browser. It works by permitting the creation of elements in web pages which use local processing to enhance the appearance of web pages and interact with the user. In order to do this, the browser provides, in the form of a specialised Java virtual machine, an execution environment which is carefully crafted to ensure there can be no risk to the local host from the actions of an applet [5, 8, 16]. Because the activity of the applet is constrained and contained, there is no risk to browser or the hosting machine associated with running the code. We realised that we could use the same mechanism to implement the nodes of a computational grid. Any machine with a Java enabled web browser could provide processing capability to a grid system if the tasks were distributed as applets embedded within a web page. Such a grid does not need its own communications infrastructure as it could use the mechanisms already in place for use by the web browser, nor is there any need to establish relationships of trust between the parties involved as the activities of applets are constrained to acceptable limits by the execution environment provided by the browser.

Since virtually any connected machine will have a suitable browser installed, virtually any connected machine worldwide could take part in such a grid as a computational node without any software installation or security implications.

2.1 How M-Grid Works in Outline

M-grid uses applet technology to distribute work to client machines. Many of the usual objections and problems associated with joining a grid are avoided because, there is no need to install software on the target machine. M-grid uses the existing web browser. Problems of security don't arise either since applets execute within the carefully constrained environment provided by the browser which is designed to protect their hosts from errant behaviour by applets. Hence, there is no need for the processing node to establish a trusting relationship with the supplier of the task or its ultimate source since the "sandbox" in which the applet executes protects the host from errant behaviour by an applet (even if it is malicious).

The only code required to create M-grid is in the provision of the coordinator role which supplies the mechanism for clients to inject their tasks into the system, distributes them to the nodes and returns the results. This is achieved using two web pages:

- A job distribution page. Anyone willing to supply processing capability to the grid does so by opening the job distribution page in a java enabled browser.
- A job submission page. Those with tasks to be performed use a form on this page to upload them and await the results.

3 Setting Up M-Grid

Since all that is required to become a processing node of M-grid is to open the job request page in a browser and clients need only to submit their tasks to the job submission page, there is no setup for these beyond directing a browser to the appropriate web page. This is unlikely to be difficult for even the most naïve of computer users. Such setup effort as there is for M-grid is in the creation of the coordinator web pages and their publication for which a machine running suitable web server software is required.

Today, most people who might be interested to setup their own M-grid will have access to a suitable web server and resources which would enable them to construct and maintain simple web pages, even if they don't do it themselves. The only complexity in the M-grid web pages is the code embedded within them which elicits tasks from clients and distributes them to the nodes. The manner in which this code is written is dependent on the capabilities of the web server software being used. For our first implementation, we have used Active Server Pages (ASP) which is provided by Microsoft in their Internet Information Server (IIS) [4].

However, we have already generated suitable code so the task of generating these pages may be reduced to including the ASP code in an appropriate web page, or adding any desired presentation to our "vanilla" M-grid files. Creating these pages this way requires no more than the most elementary skills in web page creation. With the web pages in place, the M-grid can commence operation as soon as the first clients and nodes access the pages.

4 Joining and Using M-grid

Since there is no software to install and no security or trust relationships to establish between the various elements of M-grid, joining is trivial. Almost any machine with a web browser able to view the M-grid web pages can take part.

Becoming a node offering to perform processing couldn't be easier. All that is required is to navigate a Java enabled browser to the job distribution web page and leave it. Any browser able to host an applet is suitable regardless of the underlying hardware or operating system. The page is created dynamically and displays some information about M-grid. From time to time the server will insert an applet into the page which has a computational task encoded within it. Once complete the applet returns its results (and disappears). There is nothing for the machine owner/operator to do. To remove the machine from the grid, all that is necessary is to close the browser or redirect it to another page.

The user with a job to be processed by M-grid only needs a browser able to access the form on the job submission page. Once a job has been uploaded to the server us-

ing this form, M-grid passes the task out as an applet to an available node. The results are collected and returned to the user in due course. An enhancement which we are already implementing is to permit the user uploading a job to leave the job submission page as soon as the upload is completed instead of waiting for the outcome. They can then return to the job submission page later and look up the result of their computation.

5 M-Grid in More Detail

Since the clients and computational nodes in M-grid use nothing more than standard features available in popular web browsers, there is little to say about these elements of the system. However, there are a number of considerations concerning the coordinating role.

The coordinating role runs on a machine which hosts the M-grid web pages. The server needs to collect and store jobs pending distribution to a node, arrange for each job to be sent (in due course) to just one node and to manage the results that the nodes return. This can only be achieved with a server which has the capability to generate dynamic content in the pages it supplies. These types of feature are typically offered as a scripting language and are available from all but the most minimal web servers [3, 4, 6]. Any of these could be used to create the M-grid pages. There is no one dominant technology which is universally available. Therefore, we shall need to develop different versions of the M-grid coordinating pages for each of the competing systems. For the first implementation, have used ASP which is available on Microsoft web servers (Internet Information Server, IIS and Personal Web Server, PWS). One of these servers is either included with or available as a free download for most recent versions of Windows. We felt that few potential users of M-grid would have difficulty gaining access to a machine running an ASP enabled web server. In due course, we shall implement further versions using alternative technologies such as JSP/Tomcat [1, 3] so that the M-grid coordinator pages may be hosted on other popular web serving software.

As indicated above, the coordinating role in M-grid comprises a small number of web pages published using a web server on the coordinating machine. Computational nodes set a browser to view the jobRequest page and receive their jobs as applets embedded in this page. The coordinating role achieves its objective of getting each job it receives executed by manipulating the contents of the jobRequest page so that each job is distributed to just one of the available nodes. In order to do this, the applet element to the jobRequest page is generated dynamically by the server each time it receives a request for the page.

The lifecycle of a job by M-grid has four visible stages: the two halves of the interaction with the job submission page by the client and the two halves of the interaction with the job execution page at the processing node:

1. The client submits a job via jobSubmit page.
2. The client (eventually) collects results using the jobCollectResults page.
3. A node requests a job via jobRequest page.
4. The node submits the results from a job via the jobSubmitResults page.

Let us first examine how a task is submitted to M-grid. This is depicted in Figure 1. The client with a task which they require evaluated directs their browser to the jobSubmit.asp page where they insert the details required into the form, including the location of their applet class file. On pressing the upload button the task (encapsulated in an applet class file, see later) and associated details are uploaded to the server and stored in a collection of tasks awaiting processing.

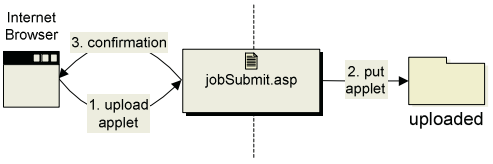


Fig. 1. Submitting an applet job to M-grid

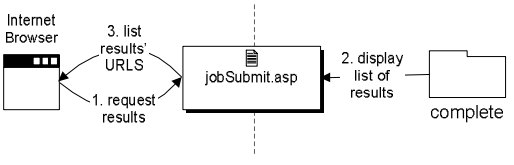


Fig. 2. Collecting results from a task

The second part of the client's interaction with the server is to collect their results. In the prototype implementation of M-grid, the user had to wait until computation was completed at which point the output was displayed in a box on the jobSubmit page. This has been amended so that the user need no longer wait for the outcome of their calculation, although they may if they wish. A returning user wishing to collect results is presented with a list of completed tasks. Clicking on any of these displays their output. This operation is shown in Figure 2. In order to keep M-grid as simple as possible, it has no security. Anyone accessing the system would be able to view the results of any task completed by the system. (Of course a user concerned to keep their work secret could submit tasks which encrypt their results.)

The remaining two visible pieces to the job's lifecycle concern the distribution of the job to a processing node and the subsequent return of the results to the coordinator. The first of these operations is illustrated in Figure 3. Along with some static content, the jobRequest.asp page includes some content which is generated dynamically. In the absence of outstanding jobs, this dynamic element is a simple message to say there are no jobs available. If there is a task waiting, it is inserted into the page as an applet. The task is then moved from the collection of uploaded tasks to a collection of tasks in progress.

Once the applet has completed, its results are uploaded to the server (using another ASP page) as illustrated in Figure 4. The task and its results are stored in a collection of completed tasks.

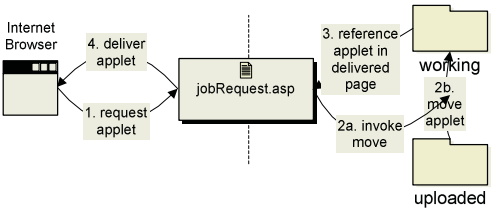


Fig. 3. Obtaining a task to perform

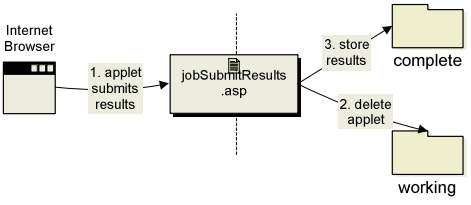


Fig. 4. Returning results

A feature of the jobRequest page is that it causes browsers to refresh the page periodically to check for new tasks so long as they are not executing an applet.

6 Issues and Future Work

M-grid has limitations. Some are a necessary consequence of the way that M-grid operates but we plan to address others. For example, since M-grid distributes tasks to nodes as applets, these tasks are restricted by the constraints which apply to applets which include being forbidden to use the local file system or to establish communications other than with the host from which the applet was downloaded.

Other limitations arise from our present implementation. One is that, with its current reliance on ASP in the coordinator web pages means that these need to be hosted on a Microsoft web server. Since Microsoft software is so widely available, we felt using ASP for the coordinator role is not a severe restriction. Nevertheless, we intend to create alternative versions of these pages which are suitable for deployment on other widely available web servers.

A further concern is the constraints which M-grid places on the tasks which clients can upload to M-grid. In its present form, a task to be executed by M-grid must be uploaded as an applet contained in a single class file and the applet must include some M-grid specific code (associated with the return of results). This requires the potential user to be able to embed their task into an applet along with the M-grid specific code. This sounds like a considerable imposition onto the user, but in fact the users of any computational grid system have to submit their tasks in the form of a program compatible with the grid framework.

We are investigating how we can eliminate the need for the M-grid specific code by either recompiling user supplied code after applying a source code transformation or providing a wrapper around the users' code. Of the restrictions to what our users are permitted to do, these arise from the nature of applets and the environment in which they execute. For example, applets are forbidden access to the local file system. Whilst some of these are inconvenient to clients creating tasks, they cannot easily be relaxed. They represent the price we pay for being able to dispatch our jobs to other machines without formality.

One final issue which we have yet to investigate fully is performance. The most obvious reason for a user to pass a job on to a computational grid is because they wish to take advantage of the processing power available from the grid. It seems reasonable to expect that the execution environment in which applets operate, in which their actions are constantly monitored to ensure they don't abuse their privilege of execution by breaching the conditions on which it is granted, would place a job executing as an applet at a performance disadvantage compared with an unrestrained application running in an open environment. However, our results to date suggest that this performance penalty is modest and quite acceptable in this context.

7 Conclusion

There is increasing interest in grid computing and there are many potential users of these technologies. However, existing grid software requires considerable installation and configuration effort, including placing software onto each machine which is to take part in the grid. Aside from any difficulty which may arise in the installation and configuration of the machines to be used, this raises issues in many environments (even if the software is to be installed on dedicated machines) and permissions need to be obtained before installation can commence.

Issues of security are usually addressed by the use of certification and trust techniques. These are crafted to ensure that so far as is possible, malicious tasks are not accepted, and assure system owners that they will not suffer harm by joining the grid and executing jobs. They also attempt to permit identification of responsibility if things go wrong. However, not only is establishing these trusting relationships complex and time consuming to set up, ultimately they still require machine owners to accept a degree of risk. By using applets, which execute in a protected environment within a web browser, M-grid is able to operate without needing any of the usual software and security infrastructure.

M-grid provides a means for potential uses to experiment with grid technologies by allowing them to set up a computational grid quickly and easily with an absolute minimum of formality, software installation and time and effort from users. It requires nothing in the way of software installation on the computational nodes or the clients and on the coordinating machine all that is needed is a web server and permission to publish web pages. Of course there is a price to pay for this convenience and this comes in the form of restrictions on the tasks which M-grid can perform to those which an applet can perform when executing within the controlled environment provided by a web browser.

References

1. The Apache Jakarta Tomcat 5.5 Servlet/JSP Container. See: http://jakarta.apache.org/tomcat/tomcat-5.5-doc/index.html (2004)
2. Altair Engineering Inc.: Portable Batch System. See: http://www.openpbs.org/ (2004)
3. Bergsten, H.: JavaServer Pages. O'Reilly and Associates (2003)
4. Buser, D., Kauffman, J., Llibre, J.T., Francis, B., Sussman, D., Ullman, C., Duckett, J.: Beginning Active Server Pages 3.0. Wiley Publishing, Inc. (2003)
5. Chen, E.: Poison Java. IEEE Spectrum. Vol. 36 (1999) 38-43
6. Converse, T.: PHP and MySQL Bible. John Wiley & Sons Inc (2004)
7. Erwin, D.W., Snelling, D.F.: UNICORE: A Grid Computing Environment. Lecture Notes in Computer Science, Vol. 2150. Springer-Verlag (2001) 825-839
8. Flanagan, D.: Java in a Nutshell. O'Reilly and Associates, (2002)
9. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Int. J. Supercomputer Applications and High Performance Computing, Vol. 11 (1997) 115-128
10. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scaleable Virtual Organization. International Journal of Supercomputer Applications and High Performance Computing, Vol. 15 (2001) 200-222
11. Frey, J., Tannenbaum, T., Livney, M., Foster, I., Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. J. Cluster Computing, Vol. 5 (2002) 237-246
12. Grandison, T., Sloman, M.: A survey of Trust in Internet Applications. IEEE Communications Surveys & Tutorials Vol. 3 (2000)
13. Litzkow, M., Livny, M.: Experience with the Condor Distributed Batch System. IEEE Workshop on Experimental Distributed Systems, Huntsville, AL (1990) 97-101
14. Livny, M., Basney, J., Raman, R., Tannenbaum, T.: Mechanisms for High Throughput Computing. SPEEDUP Journal Vol. 11 (1997) 36-40
15. Gridsystems S.A.: Overview to InnerGrid. See: http://www.gridsystems.com/pdf/IGIntro.pdf (2003)
16. Sun Microsystems, Inc.: Sun Microsystems, Java Technology. See: http://www.java.sun.com/ (2004)
17. Sunderam, V., Kurzniec, D.: Lightweight Self-organizing Frameworks for Metacomputing. 11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland (2002) 113-122

GLIDE: A Grid-Based Light-Weight Infrastructure for Data-Intensive Environments

Chris A. Mattmann[1,2], Sam Malek[2], Nels Beckman[2], Marija Mikic-Rakic[2], Nenad Medvidovic[2], and Daniel J. Crichton[1]

[1] Jet Propulsion Laboratory, 4800 Oak Grove Drive, M/S 171-264, Pasadena, CA 91109, USA
{chris.mattmann, dan.crichton}@jpl.nasa.gov
[2] University of Southern California, Los Angeles, CA 90089, USA
{mattmann, malek, nbeckman, marija,neno}@usc.edu

Abstract. The promise of the grid is that it will enable public access and sharing of immense amounts of computational and data resources among dynamic coalitions of individuals and institutions. However, the current grid solutions make several limiting assumptions that curtail their widespread adoption in the emerging decentralized, resource constrained, embedded, autonomic, and mobile (DREAM) environments: they are designed primarily for highly complex scientific problems, and therefore require powerful hardware and reliable network connectivity; additionally, they provide no application design support to grid users (e.g., scientists). To address these limitations, we present GLIDE, a prototype light-weight, data-intensive middleware infrastructure that enables access to the robust data and computational power of the grid on DREAM platforms. We illustrate GLIDE on an example file sharing application. We discuss our early experience with GLIDE and present a set of open research questions.

1 Introduction

One of the most exciting and promising technologies in modern computing is the grid [4,6]. Grid computing connects dynamic collections of individuals, institutions, and resources to create *virtual* organizations, which support sharing, discovery, transformation, and distribution of data and computational resources. Distributed workflow, massive parallel computation, and knowledge discovery are only some of the applications of the grid. Grid applications involve large numbers of distributed devices executing large numbers of computational and data components. As such, they require techniques and tools for supporting their design, implementation, and dynamic evolution.

Current *grid* technologies provide extensive support for describing, modelling, discovering, and retrieving data and computational resources. Unfortunately, they are predominantly implemented using middleware infrastructures that leverage both heavyweight and computationally intensive protocols and objects [3]. As such, current grid software systems are not readily applicable to the domain of decentralized, resource constrained, embedded, autonomic, and mobile (DREAM) environments. Existing grid technologies also lack native support for systematic application design, implementation, and evolution. Finally, the development, deployment, and runtime adaptation support for the grid is ad-hoc: shell scripts abound, makefiles are the common construction and deployment tool, and adaptation is usually handled by restarting the entire system.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 68 – 77, 2005.

Given the central role that software architectures have played in engineering large-scale distributed systems [12], we hypothesize that their importance will only grow in the even more complex (grid-enabled) DREAM environments. This is corroborated by the preliminary results from several recent studies of software architectural issues in embedded, mobile, and ubiquitous systems [10,14,19]. In order for architectural models to be truly useful in any development setting, they must be accompanied by support for their implementation [8]. This is particularly important for the DREAM environments: these systems will be highly distributed, decentralized, mobile, and long-lived, increasing the risk of architectural drift [12] unless there is a clear relationship between the architecture and its implementation. To address these issues, several light-weight *software architecture*-based solutions [10,14] supporting the design, implementation, and evolution of software systems in DREAM environments have recently emerged. However, these solutions are still not directly supporting the grid: they have not focused on, and uniformly lack facilities for, resource and data description, search, and retrieval.

A recent focus of our work has been on addressing the limitations of the grid by bridging the two approaches described above. Specifically, we have drawn upon our previous experience in developing the OODT data grid middleware [2,8] along with our experience in developing the Prism-MW middleware for resource constrained devices [9,10], to arrive at GLIDE, a grid-based, lightweight infrastructure for data-intensive environments. GLIDE was built with a focus on addressing both the resource limitations and lack of systematic application development support of the current grid technologies. GLIDE strives to marry the benefits of Prism-MW (architecture-based development, efficiency, and scalability) with those of OODT (resource description, discovery, and retrieval). We have performed a preliminary evaluation of GLIDE using a series of benchmarks, and have successfully tested it by creating a mobile media sharing application which allows users to share, describe and locate mp3 files on a set of distributed PDAs. While this work is still in its early stages, our initial results have been promising and have pointed to several avenues of future work.

The rest of this paper is organized as follows. Section 2 describes the existing grid middleware infrastructures and presents an overview of Prism-MW. Section 3 describes the design, implementation, and evaluation of GLIDE and is illustrated using an example MP3 sharing application. The paper concludes with an overview of future work.

2 Background and Related Work

GLIDE has been inspired by a set of related projects along with our own existing work in three areas: computational and data grids, light-weight middleware and protocols, and implementation support for software architectures. In this section, we first briefly overview existing grid solutions, and their most obvious limitations that motivated GLIDE. We then describe OODT, the grid technology used by NASA and the National Cancer Institute, along with other representative approaches to large-scale data sharing. Finally, we summarize Prism-MW, a light-weight middleware platform that explicitly focuses on implementation-level support for software architectures in DREAM environments; we also briefly overview a cross-section of representative light-weight middleware platforms.

2.1 Computational Grid Technologies

Globus [4,6] is an open-source middleware framework for constructing and deploying grid-based software systems, which has become the *de facto* standard grid toolkit. Glo-

bus realizes the basic goal of the grid: the establishment of virtual organizations sharing computational, data, metadata, and security resources. However, Globus lacks several development features that would ease its adoption and use across a more widespread family of software systems and environments. These features include (1) architecture-based development, (2) deployment and evolution support (currently makefiles and shell-scripts are the standard build tools) and (3) lightweight implementation substrates.

In addition to Globus, several other grid technologies have emerged recently. Alchemi [1] is based on the Microsoft .NET platform and allows developers to aggregate the processing power of many computers into virtual computers. Alchemi is designed for deployment on personal computers: computation cycles are only shared when the computer is idle. JXTA [7] is a framework for developing distributed applications based on a peer-to-peer topology. Its layered architecture provides abstractions of low-level protocols along with services such as host discovery, data sharing, and security.

2.2 Data Grid Technologies

GLIDE is directly motivated by our own work in the area of data-grids, specifically on the Object Oriented Data Technology (OODT) system [2]. We have adopted an architecture-centric approach in OODT [8], in pursuit of supporting distribution, processing, query, discovery, and integration of heterogeneous data located in distributed data sources. Additionally, OODT provides methods for resource description and discovery based on the ISO-11179 data model standard [17], along with the Dublin Core standard for the specification and standardization of data elements [18].

There are several other technologies for large-scale data sharing. Grid Data Farm[15] project is a parallel file system created for researchers in the field of high energy acceleration. Its goal is to federate extremely large numbers of file systems on local PCs and, at the same time, to manage the file replication across those systems, thus creating a single global file system. Similar to OODT, the SDSC Storage Resource Broker [13] is a middleware that provides access to large numbers of heterogeneous data sources. Its query services attempt to retrieve files based on logical information rather than file name or location, in much the same way that OODT maintains profile data.

2.3 Prism-MW

Prism-MW [10] is a middleware platform that provides explicit implementation-level support for software architectures. The key software architectural constructs are *components* (units of computation within a software system), *connectors* (interaction facilities between components such as local or remote method calls, shared variables, message multicast, and so on), and *configurations* (rules governing the arrangements of components and connectors) [12]. The top-left diagram in Figure 1 shows the class design view of Prism-MW's core. Brick is an abstract class that encapsulates common features of its subclasses (*Architecture, Component, and Connector*). The *Architecture* class records the configuration of its components and connectors, and provides facilities for their addition, removal, and reconnection, possibly at system runtime. A distributed application is implemented as a set of interacting *Architecture* objects, communicating via *DistributionConnectors* across process or machine boundaries. *Components* in an architecture communicate by exchanging Events, which are routed by *Connectors*. Finally, Prism-MW associates the *IScaffold* interface with every *Brick*. Scaffolds are used to schedule and dispatch events using a

pool of threads in a decoupled manner. *IScaffold* also directly aids architectural self-awareness by allowing the runtime probing of a *Brick's* behavior.

Prism-MW enables several desired features of GLIDE. First, it provides the needed low-level middleware services for use in DREAM environments, including decentralization, concurrency, distribution, programming language abstraction, and data marshalling and unmarshalling. Second, unlike the support in current grid-based middleware systems (including OODT), Prism-MW enables the definition and (re)use of architectural styles, thereby providing design guidelines and facilitating reuse of designs across families of DREAM systems. Third, Prism-DE [9], an architecture-based (re-)deployment environment that utilizes Prism-MW, can be extended to aid GLIDE users in constructing, deploying, and evolving grid-based DREAM systems.

A number of additional middleware technologies exist that support either architectural design or mobile and resource constrained computation, but rarely both [10]. An example of the former is Enterprise Java Beans, a popular commercial technology for creating distributed Java applications. An example of the latter is XMIDDLE [16], an XML-based data sharing framework targeted at mobile environments.

3 Arriving at GLIDE

GLIDE is a hybrid grid middleware which combines the salient properties of Prism-MW and core services of the grid, with the goal of extending the reach of the grid beyond super-computing and desktop-or-better platforms to the realm of DREAM environments. To this end, the myriad of heterogeneous data (music files, images, science data, accounting documents, and so on) and computational (web services, scientific computing testbeds, and so on) resources made available by heavy-weight grids can also be made available on their mobile counterparts. Thus, mobile grids enabled by GLIDE have the potential to be both *data-intensive*, requiring the system to provide rich metadata describing the abundant resources (and subsequently deliver and retrieve representatively large amounts of them), as well as *computationally-intensive*, focused on discovering and utilizing data, systems, authorization, and access privileges to enable complex, distributed processing and workflow.

Existing grid solutions such as Globus and OODT take a completely agnostic approach to the amount of hardware, memory, and network resources available for deploying, executing, and evolving a grid-based software system. These technologies consider the system's architectural design to be outside their scope. In addition, they also fail to provide sufficient development-level support for building, deploying, and evolving software applications. A solution that overcomes these limitations is needed to realize the widely stated vision of "data and computation everywhere". By implementing the core grid components of OODT using Prism-MW, we believe to have created an effective prototype platform for investigating and addressing these limitations.

3.1 GLIDE's Design

We specialized Prism-MW to implement the core components of GLIDE shown in Figure 1. Our first objective was to retain the key properties and services of Prism- MW and provide basic grid services (such as resource discovery and description, search and retrieval) across dynamic and mobile virtual organizations. Additionally, we desired GLIDE to support architecture-based design, implementation, deployment, and evolution of data-intensive grid applications in DREAM environments. Finally, we

desired that GLIDE at least partially interoperate with a heavy-weight grid counterpart: because of our prior experience with the OODT middleware, it seemed the most appropriate choice; indeed, OODT directly influenced our design of the key grid services provided by GLIDE. Below we describe GLIDE's architecture in light of these objectives.

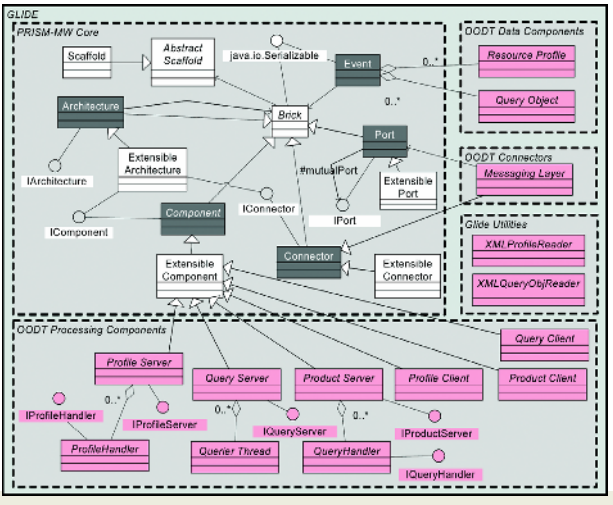


Fig. 1. Architecture diagram of GLIDE showing its Prism-MW and OODT foundation

Inspired by OODT's architecture, GLIDE's *Data Components* include the *Resource Profile*, a data structure which describes the location and classification of a resource available within a grid-based software system. Resources include data granules (such as a File), data-producing software systems (including the below described profile servers, product servers, query servers, and so on), computation-providing software systems, and resource profiles themselves. Resource profiles may contain additional resource-describing metadata [2]. The *Query Object* is a data structure which contains a query expression. A query expression assigns values to a predefined set of data elements that describe resources of interest to the user and a collection of obtained results.

Again, inspired by OODT's architecture, GLIDE's *Processing Components* include *Product Servers*, which are responsible for abstracting heterogeneous software interfaces to data sources (such as an SQL interface to a database, a File System interface to a set of images, an HTTP interface to a set of web pages, and so on) into a single interface that supports querying for retrieval of data and computational resources. Users query product servers using the query object data structure. *Product Clients* connect and send queries (via a query object) to product servers. A query results in either data retrieval or use of a remote computational resource. *Profile Servers* generate and deliver metadata [2] in the form of resource profile data structures, which are used for making informed decisions regarding the type and location of resources that satisfy given criteria. *Profile Clients* connect and send queries to profile servers. After sending a query, a profile client waits for the profile server to send back any resource profiles that satisfy the query. *Query Servers* accept query objects, and then use profile servers to determine the available data or computational resources that satisfy the user's query. Once all the resources have been collected, and processing has occurred, the data and processing results are returned (in the form of the result list of a query object) to the originating user. *Query Clients* connect to query servers, issue queries, and retrieve query objects with populated data results. GLIDE contains one *software connector*. The *Messaging Layer* connector is a data bus which marshals resource profiles and query objects between GLIDE client and server components.

Each GLIDE processing component was implemented by subclassing Prism-MW's *ExtensibleComponent* class, using the asynchronous mode of operation. Asynchronous interaction directly resulted in lower coupling among GLIDE's processing components. For example the dependency relationships between GLIDE's *Client* and *Server* processing components, which existed in OODT, are removed. GLIDE's components use Prism-MW's *Events* to exchange messages. GLIDE data components are sent between processing components by encapsulating them as parameters in Prism-MW *Events*. Leveraging Prism-MW's *Events* to send and receive different types of data enables homogenous interaction among the processing components.

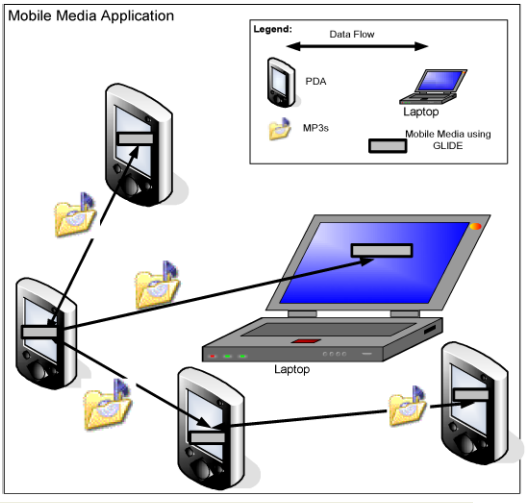


Fig. 2. Mobile Media Sharing Application

We found OODT's connectors not to be suitable for DREAM environments because of their heavy-weight (they are implemented using middleware such as RMI and CORBA). Furthermore, they only support synchronous interaction, which is difficult to effect in highly decentralized and mobile systems characterized by unreliable network links. To this end, we have leveraged Prism-MW's asynchronous connectors to implement the messaging layer class in GLIDE. GLIDE's connector leverages Prism-MW's port objects that allow easy addition or removal of TCP/IP connections, thus allowing the system's topology to be adapted at runtime. GLIDE's connector also implements event filtering such that only the requesting client receives responses from the server.

The high degree of decoupling among GLIDE messaging layer components directly aids easy dynamic adaptation, the lack of which is a key limitation in current grid systems. Ability to easily adapt a system's software architecture is an important property missing in OODT that can be leveraged to improve the system's functionality, scalability, availability, latency, and so on. For example, our recent studies [10] have shownthat the availability and latency of software systems in DREAM environments can be improved significantly via dynamic adaptation.

Finally, to support interoperability of GLIDE with OODT, we provide two additional utility classes: *XMLProfileReader* and *XMLQueryObjReader* parse an XML representation of a resource profile and query object data structure, respectively. Due to space limitations, we cannot provide a detailed example of the XML Profile structure here, but its full treatment can be found in [2]. Each string is parsed into a GLIDE data object. Similarly, resource profiles and query objects can be serialized into XML. Thus, the level of interoperability with OODT is at the level of resource description and retrieval, and currently resource profiles and data can be exchanged

between GLIDE and OODT. As part of our current work, we are investigating the Web Services Resource Framework (WS-RF) as a means of enabling interoperability between GLIDE and Globus, which looks to use WS-RF in its latest release.

3.2 Sample Application Using GLIDE

In order to evaluate the feasibility of GLIDE, we designed and implemented a *M*obile *M*edia *S*haring application (MMS), shown in Figure 2. MMS allows a user to query, search, locate, and retrieve MP3 resources across a set of mobile, distributed, resource-constrained devices. Users query mobile media servers for MP3 files by specifying values for genre and quality of the MP3 (described below), and if found, the MP3s are streamed asychronously to the requesting mobile media client.

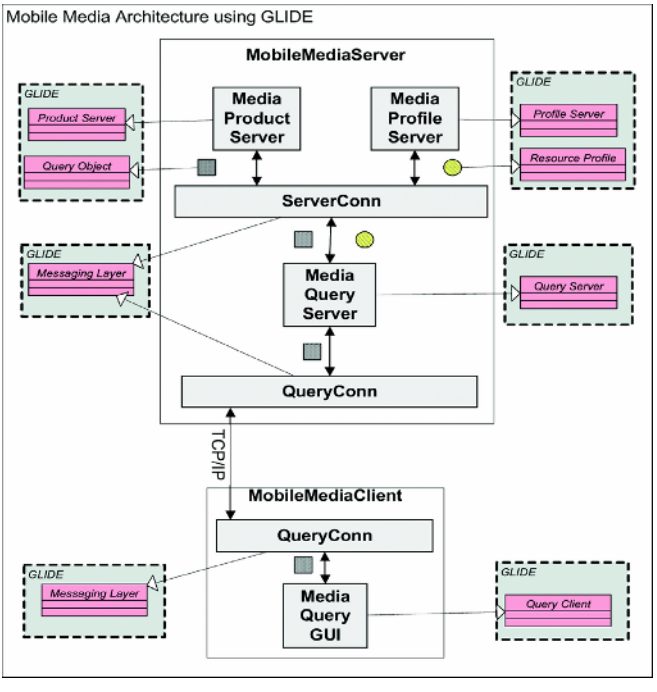


Fig. 3. Mobile Media Application Architecture

Figure 3 shows the overall distributed architecture of the MMS application. A mobile device can act as a server, a client, or both. *MobileMediaServer* and *MobileMediaClient* correspond to the parts of the application that are running on the server and the client devices.

MobileMediaClient contains a single component called *MediaQueryGUI*, which provides a GUI for creating MP3 queries. MP3 queries use two query parameters, *MP3.Genre* (e.g., rock) and *MP3.Quality* (e.g., 192 kb/s, 128 kb/s). *MediaQueryGUI* is attached to a *QueryConn*, which is an instance of GLIDE's messaging layer connector that forwards the queries to remote servers and responses back to the clients.

MobileMediaServer is composed of three component types: *MediaQueryServer*, *MediaProductServer*, and *MediaProfileServer*. *MediaQueryServer* parses the query received from the client, retrieves the resource profiles that match the query from *MediaProfileServer*, retrieves the mp3 file(s) in which the user was interested from the *MediaProductServer*, and sends the MP3 file(s) back to the client.

The MMS application helps to illustrate different aspects of GLIDE: it has been designed and implemented by leveraging most of GLIDE's processing and data components and its messaging layer connector, and has been deployed on DREAM devices. In the next section we evaluate GLIDE using MMS as an example.

3.3 Evaluation

In this section we evaluate GLIDE along the two dimensions outlined in the Introduction: (1) its support for architecture-based Development and deployment, and (2) its suitability for DREAM environments.

3.3.1 Architecture-Based Development and Deployment Support

GLIDE inherits architecture-based development and deployment capabilities, including style awareness, from Prism-MW and deployment support, from PRISM-DE. Unlike most existing grid middleware solutions (e.g. OODT), which provide support for either peer-to-peer or client-server styles, GLIDE does not impose any particular (possibly ill-suited) architectural style on the developers of a grid-based application. As a proof of concept, we have implemented several variations of the MMS application in different architectural styles including client-server, layered client-server, peer-to-peer, and C2 [20]. The variations of MMS leveraged existing support for these styles and were created with minimal effort. For example, changing MMS from client-server to peer-to-peer required addition of three components and a connector on the server side, and one component and one connector on the client side. Figure 4 shows the peer-to-peer variant of MMS.

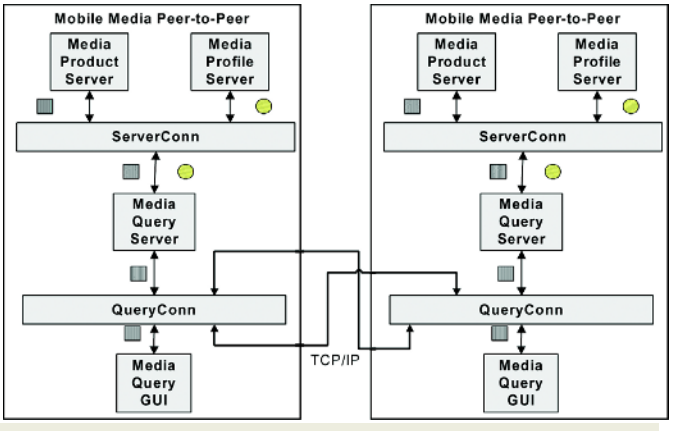


Fig. 4. Peer-to-peer variation of the Mobile Media application

3.3.2 DREAM Support

Resource scarcity poses the greatest challenge to any grid solution for DREAM environments. We have leveraged Prism-MW's efficient implementation of architectural constructs [10] along with the following techniques to improve GLIDE's performance and minimize the effect of the computing environment's heterogeneity: (1) MinML [11], a lightweight XML parser, to parse the resource profiles and query object data structures; (2) W3C's Jigsaw Web Server Base64 Encoding Library [5] to compress (at the product server end) and uncompress (at the product client end) the exchanged data; (3) Filtering inside the Messaging Layer to ensure event delivery only to the interested parties, thus minimizing propagation of events with large data loads (e.g., MP3 files). Specifically, GLIDE tags outgoing request events from a client with a unique ID, which is later used to route the replies appropriately; and (4) Incremental data exchange via numbered data segments for cases when the reliability of connectivity and network bandwidth prevent atomic exchange of large amounts of data.

Table 1. Memory footprint of MobileMediaServer and MobileMediaClient in GLIDE

MobileMediaServer	Java Packages	# Live Objects	Total Size (bytes)
Java	java.lang	36	2016
GLIDE's Implementation of OODT components	glide.product	1	24
	glide.profile	1	24
	glide.query	1	32
	glide.queryparser	1	160
	glide.structs	8	232
Application	mobilemedia.product.handlers	1	32
	mobilemedia.profile.handlers	1	8
Prism-MW	glide.prism.core	26	1744
	glide.prism.extensions.port	1	40
	glide.prism.extensions.port.distribution	4	216
	glide.prism.handler	2	32
	glide.prism.util	18	1200
Total size			***5760***

MobileMediaClient			
Java	java.lang	28	1568
GLIDE's implementation of OODT components	glide.structs	7	208
Application	mobilemedia	2	384
Prism-MW	glide.prism.core	18	1304
	glide.prism.extensions.port	1	40
	glide.prism.extensions.port.distribution	3	136
	glide.prism.handler	1	16
	glide.prism.util	7	480
Total size			***4136***

As an illustration of GLIDE's efficiency, Table 1 shows the memory footprint of *MobileMediaServer's* and *MobileMediaClient's* implementation in GLIDE. The total size of the *MobileMediaServer* was 5.7KB and *MobileMediaClient* was 4.1KB, which is two orders of magnitude smaller than their implementation in OODT (707KB and 280KB, respectively). The memory overhead introduced by GLIDE on the client and server devices was under 4KB.

4 Conclusions and Future Work

This paper has presented the motivation for and prototype implementation of a grid platform for decentralized, resource constrained, embedded, autonomic, and mobile (DREAM) environments. Although the results of our work to date are promising, a number of pertinent issues remain unexplored. Future work will focus on (1) extending GLIDE to provide a set of meta-level services, including monitoring of data and meta-

data; and (2) addressing the resource replication issue in grid applications. We believe that GLIDE will afford us an effective platform for investigating this rich research area.

Acknowledgements. This material is based upon work supported by the National Science Foundation under Grant Numbers CCR-9985441 and ITR-0312780. Effort also supported by the Jet Propulsion Laboratory, managed by the California Institute of Technology.

References

1. Alchemi .NET Grid Computing Framework. http://www.alchemi.net/doc/0_6_1/index.html
2. D. J. Crichton, J. S. Hughes, and S. Kelly. A Science Data System Architecture for Information Retrieval. in *Clustering and Information Retrieval*. W. Wu, H. Xiong, and S. Shekhar, Eds.: Kluwer Academic Publishers, 2003, pp. 261-298.
3. N. Davies, A. Friday and O. Storz. Exploring the Grid's potential for ubiquitous computing. *IEEE Pervasive Computing*, Vol 3. No. 2, April-June, 2004, pp.74-75.
4. I. Foster et al. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Globus Research*, Work-in-Progress 2002.
5. Jigsaw Overview. http://www.w3.org/Jigsaw/.
6. C. Kesselman, I. Foster, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of Supercomputing Applications*, pp. 1-25, 2001.
7. N. Maibaum, T. Mundt. JXTA: A Technology Facilitating Mobile Peer-to-Peer Networks. *MobiWac 2002*. Fort Worth, TX, October 2002.
8. C. Mattmann et al. Software Architecture for Large-scale, Distributed, Data-Intensive Systems. 4th *Working IEEE/IFIP Conference on Software Architecture*, Oslo, Norway, 2004.
9. M. Mikic-Rakic and N. Medvidovic. Architecture-Level Support for Software Component Deployment in Resource Constrained Environments, *1st International IFIP/ACM Working Conference on Component Deployment*, Berlin, Germany, 2002.
10. M. Mikic-Rakic and N. Medvidovic. Adaptable Architectural Middleware for Programming-in-the-Small-and-Many. *ACM/IFIP/USENIX Middleware Conference*, Rio De Janeiro, Brazil, 2003.
11. MinML A Minimal XML parser. http://www.wilson.co.uk/xml/minml.htm.
12. D. E. Perry and A. L. Wolf. Foundations for the Study of Software Architecture, *ACM SIGSOFT Software Engineering Notes (SEN)*, vol. 17, 1992.
13. A. Rajasekar, M. Wan, R. Moore. MySRB and SRB - Components of a Data Grid. *High Performance Distributed Computing (HPDC-11)*. Edinburgh, UK, July 2002.
14. J. P. Sousa and D. Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. *3rd Working IEEE/IFIP Conference on Software Architecture (WICSA-2002)*, Montreal, Canada, 2002, pp. 29-43.
15. O Tatebe et. al. The Second Trans-Pacific Grid Datafarm Testbed and Experiments for SC2003. *2004 International Symposium on Applications and the Internet*, January 2004, Tokyo, Japan.
16. S. Zachariadis et. al. XMIDDLE: Information Sharing Middleware for a Mobile Environment. *ICSE 2002*, Orlando, FL, May 2002.
17. ISO/IEC, Framework for the Specification and Standardization of Data Elements, Geneva, 1999.
18. DCMI, Dublin Core Metadata Element Set, Version 1.1: Reference Description, 1999
19. L. McKnight, J. Howison, and S. Bradner. Wireless Grids: Distributed Resource Sharing by Mobile, Nomadic, and Fixed Devices. *IEEE Internet Computing*, pp. 24-31, July/August 2004.
20. R. N. Taylor, N. Medvidovic, et al. A Component-and-Message-Based Architectural Style for GUI Software. *IEEE Transactions on Software Engineering*, Vol. 22, No. 6, pp. 390-406, June 1996.

HotGrid: Graduated Access to Grid-Based Science Gateways

Roy Williams, Conrad Steenberg, and Julian Bunn

California Institute of Technology, Pasadena, California, USA

Abstract. We describe the idea of a Science Gateway, an application-specific task wrapped as a web service, and some examples of these that are being implemented on the US TeraGrid cyberinfrastructure. We also describe HotGrid, a means of providing simple, immediate access to the Grid through one of these gateways, which we hope will broaden the use of the Grid, drawing in a wide community of users. The secondary purpose of HotGrid is to acclimate a science community to the concepts of certificate use. Our system provides these weakly authenticated users with immediate power to use the Grid resources for science, but without the dangerous power of running arbitrary code. We describe the implementation of these Science Gateways with the Clarens secure web server.

1 Science Gateways

For many scientists, the Grid is perceived as difficult to use. There are several reasons for this: accounts are issued in response to a proposal, which may take weeks to review. Security concerns often mean that users cannot use passwords, but rather public-key or certificate systems that have steep learning curves. Users are then faced with just a Unix prompt, which many scientists find daunting. Other challenges include difficult porting of code, arcane archival storage paradigms, the need to learn Globus, Condor, GridFTP, and so on. In return for this difficulty is the promise of access to great computational power, but this only for a small number of dedicated people who understand both Grid and science.

Conscious of these difficulties, we have devised the HotGrid concept reported on in this paper, which makes domain-specific ease of use the driving factor. We believe that the balanced, pragmatic approach outlined will not only provide useful science gateways in the medium to long term, but also strong scientific results in the short term. We will explore the construction of "science gateways" which abstract the architectural complexity of Grid resources and scientific applications implemented on the Grid, presenting their capabilities to communities ("Virtual Organizations") of possibly anonymous scientists, who need no explicit knowledge of the Grid. This approach to science gateways will provide a graduated and documented path for scientists, from anonymous user, to the weakly authenticated *HotGrid* user, through to the power user already comfortable with the Grid.

Typically, a gateway developer is a scientist who wishes to host an application and expose its use with a web form[1][2] – a simple way to describe a computation

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 78–87, 2005.

request, with input fields to allow selection of dataset, parameters, response format, etc., and an output data product returned, for example, as a URL. The developer wants simplicity and transparency: a simple tool to do a simple job. Once the web-form approach shows the service to be useful to a community, there is generally a need to progress to a script-based scheme, or to have the service functions embedded in a familiar data-analysis environment. The gateway is thus utilized in several different ways:

- Web form: the user can use the service in a documented environment optimized for teaching the why and how of the computation that the gateway offers. This interface is also excellent for determining the health and "heartbeat" service status.
- Data Analysis Environment: here the user executes a familiar data analysis program such as ROOT[8], VOStatistics[9], IDL[10], or IRAF[11]. The scientist can concetrate on the scientific goals, rather than the computation logistics. The Science Gateway service is abstracted as a remote computation service, and appears to the scientist just as any other service, except that the considerable potential power of the Grid is employed to execute that service. For weakly authenticated users, only a subset of predefined service commands are allowed, making execution of arbitrary code impossible.
- Programming interface (API): the scientist uses an interface written in a high level language that uses the standard web-service protocols (SOAP, XML-RPC) to access the service. In the case of SOAP automatic generation of the interface stubs can be foreseen thanks to the use of the WSDL (Web Services Description Language) as a service descriptor. The connection of the Data Analysis Environment (mentioned in the previous bullet) can be made using this API.
- Logging in to the Grid itself: the scientist who is a "power user" can log in to the host node of the Science Gateway and run a chosen application directly from the command line. This is the traditional way of using a centralised computing facility, and it requires the highest form of authentication due to its wide scope.

1.1 Graduated Security and HotGrid

The HotGrid scheme aims to make the Grid easy to use, yet powerful. This is achieved by gradating access semantics starting from the level of anonymous user, who can obtain a small allocation of CPU time, to the power user, who can be allocated a large quantity of resources, but who is required to write a resource allocation proposal to a committee, wait for reviews to be obtained, and finally to obtain authentication clearance.

There is a crucial difference between two kinds of authentication: the power to use a lot of computer resources, and the power to run arbitrary code. The former would, for example, allow a malicious person to execute an excessive number of tasks in the Grid. At worse, this would constitute a "denial of service" activity, which an operator or alert user would notice and have stopped.

In contrast, a malicious person able to run arbitrary code can do very serious damage very quickly, for example by deleting files, spoofing, and expanding their authentication and credentials to gain elevated access in the system and other mutually trusted systems.

Thus the HotGrid scheme is focused on allowing weakly authenticated users the ability to use powerful resources, but not allowing the execution of arbitrary code.

1.2 Levels of Authentication

The following list illustrates the graduation from none, through weak and then to strong authentication:

- A Science Gateway service runs a specific code with parameters that control it. The service is exposed via a web form that allows a small job to be executed anonymously and easily. The user might be allocated one CPU-hour in any 24-hour period.
- At the next level is the *HotGrid* certificate. The user completes a form specifying who they are and what they want to do, and obtains a certificate that allows perhaps 30 CPU-hours to be expended on jobs submitted via the Science Gateway. When the HotGrid certificate expires, the user must fill out the form again to obtain a new HotGrid certificate. (The analogy is filling in a form to get a `hotmail.com` account.)
- If the user needs more resources, a strong certificate must be obtained, from a certificate authority that makes proper checks on the user's claimed identity (eg. DOE, NASA, TeraGrid, Verisign etc.). The user will then attach that strong certificate to the HotGrid form submission, and thereby obtain more resources according to grant allocation policies, perhaps 100 CPU-hours in any given month.
- Finally, the user may apply for Grid resources in the usual way, via a proposal submitted for review (for TeraGrid accounts, see [12]) and then be allocated a power-user account, with the associated right to use, say, 1000s of CPU-hours per month. The user may also continue to use the Science Gateway, or can log in with `ssh` and use traditional job submission techniques.

Currently, most Grid projects recognize only the power user. Some projects are beginning to create anonymous gateways on non-Grid resources. We propose to work with TeraGrid management to interpolate between these with the intermediate levels in the list above, including the HotGrid level.

Initially, we will focus on two scientific application areas, high energy physics (HEP) and virtual astronomy. The communities and scientists involved in these fields are distributed across the nation and around the world. Both groups already have substantial TeraGrid allocations and are planning additional requests. Based on experience with HEP and virtual astronomy, we will widen the scope of our work to support other scientific disciplines.

1.3 Certificate Authentication

A certificate is a means of *authentication*, meaning that when it is properly presented, it can *prove* that the presenting entity is who it says it is. Certificates can be used by computers to prove their identity (*"Yes this really is caltech.edu, not a spoof"*), but in this paper we shall think of the certificate as authenticating a human. Note that a certificate does *not* specify an account or allocation of any kind of resources, but rather it is used to prove identity, so that resources owned by the owner or group of owners can be utilized. A certificate generally has a start and an end (or revocation) date, and becomes invalid after the end date.

The X.509 certificate is a digital document; when combined with a secret *passphrase* and presented to a server, it provides proof that the correct secret phrase was typed. This happens without the passphrase being stored or transmitted on an open connection, so it is very difficult to steal passphrases from a remote location. Of course it is possible that the passphrase has been compromised; perhaps the owner told a colleague, or there was spyware on a machine, or a camera watched keystrokes. If not compromised, however, then presentation of the certificate is strong evidence that the person presenting is the same as the person who thought of the passphrase. Therefore, the requirement for authentication is to make sure that a person asking for a certificate to be issued is who they say they are: this is the responsibility of a Certificate Authority (CA).

A strong CA will only issue a certificate when appropriate proof of identity is presented: passport, driver's licence, or referral by a trusted third party. We can imagine a weak CA that issues certificates with little proof. Examples abound in real life: a passport is a strong form identification since it is hard to obtain without strong personal credentials and effort, whereas a membership card at a public library is rather easy to obtain. There is a compromise between the strength of the certificate and the difficulty of obtaining it. Generally, a weak certificate can only be used in a restricted context: a fake library card can only allow an imposter to steal a few books, whereas a fake passport can be used to build a whole new life.

It is easy to set up a CA and issue certificates. So, those who accept certificates always have a list of CAs that they trust, and any new or previously unknown CA must convince a resource provider to accept their certificates, otherwise they are of no use to the certficates' owners. Part of the difficulty of using the major Grid systems (e.g. the US TeraGrid, the European Grid, the UK eScience Grid) is the burdensome procedure of obtaining a sufficiently strong certificate that is accepted by the system.

However, we hope to convince the administrators of these major Grids to accept a certain class of weak certificate: the *HotGrid* certificates. These will be obtainable quickly, simply by completing a form at a special HotGrid CA web site, with name and address and a short abstract or description of the intended purpose of the Grid work planned. Techniques for proving that a human is filling in the form, rather than a bot, will be employed. The HotGrid certificate will

be issued immediately, but its use will be restricted to a particular time period, or for executing a restricted set of tasks, and/or for using a limited amount of resources.

1.4 User Experience

Our target user in the HotGrid project is the domain scientist who knows no more about the Grid than the rhetoric from the colour supplements. This person is not interested in downloading and installing software, nor in writing a proposal. This person is primarily interested in using the power of the Grid to solve a scientific problem. We assume that they have a web browser, an internet connection, and some time to work on the problem.

The user can simply connect to a web server that runs a science gateway, such as those in the examples below. We expect each gateway to offer minimal functionality to the anonymous user, which will allow our user to try the system and become convinced that a little more effort is worthwhile.

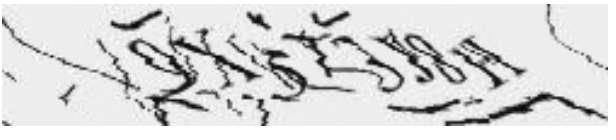


Fig. 1. An example of a CAPTCHA image (Completely Automated Public Turing test to tell Computers and Humans Apart) that can be used to ensure that HotGrid certificates cannot be obtained automatically

At this point, our user will be directed to a CA, and asked to fill out a form with their name, location, and a chosen passphrase. They will also be asked to specify which science gateway they wish to use, and for a short explanation of the envisaged purpose. In the same way as the `hotmail.com` system allocates free email accounts, the HotGrid system will prevent a "bot" from obtaining HotGrid certificates automatically by use of techniques that can differentiate human-based form entries from from computer-generated entries (see Fig. 1). The CA will create an authentication certificate for the user in response to a successful form submission. This will be returned to the user and also installed on the science gateway. We note that this certificate is very weak authentication – in fact all it shows is that a human filled in the form, but we also note that an advantage to the procedure is that domain scientists learn what a certificate is and how it works. Armed with the certificate, our user can now install it in a web browser (online help is provided by all the popular web browsers for this simple task).

With the HotGrid certificate in the browser, the user can now enjoy access to more Grid resources via the science gateway. The certificate is restricted by time and named gateway. If the user wishes to make more extensive use of the science

gateway, it can be done by obtaining a better, longer-lasting certificate, perhaps through referral by a colleague, by membership in a professional organization, or by providing stronger proof of identity. The user may also choose to obtain a power-user account on the Grid system by writing a proposal or by joining an existing trusted project.

Thus, by the use of the HotGrid scheme, we hope to engage and empower a much wider community of domain scientists in Grid-based scientific computing than would otherwise be expected.

2 Clarens

Clarens[13] is a Grid-enabled web services framework that provides a secure, high-performance foundation for constructing web services. It accepts web service requests from clients formatted as SOAP or XML-RPC messages, processes these requests, and returns responses in the same message format.

Clients range from web browsers, through simple scripts, up to large compiled applications. A minimum of software environment requirements are imposed on these clients. Clarens provides a standard application programming interface (API) that is independent of transport protocol or message format.

Two server implementations are currently provided: one using the well-known Apache web server with an embedded Python interpreter, and the other in Java with the Apache Tomcat servlet container. Clarens is agnostic towards operating systems: both Unix and Windows are supported. A set of standard services are available to clients. These include authentication, file access, shell command execution and tools for administration of security. Users with recognized certificates can be organized in a rich hierarchical structure of groups and sub-groups, and thus into virtual organizations (VO).

2.1 Clarens Security

Clients are authenticated using standard X.509 certificates, using SSL encryption, as used by web browsers when communicating with e.g. banking web sites. As an additional precaution clients can also be authenticated using so-called proxy, or temporary, certificates that are valid for a limited period of time, but which still provide cryptographically secure user authentication. These proxy certificates can be generated on the client computer or by making use of an intermediate proxy server that uses a username and password combination for authentication. Once a client is authenticated, Clarens provides access to services, refined by the use of using access control lists (ACLs) for individuals or groups of individuals to each service (e.g. task submission, access to files, permission to modify group definitions, etc.).

Clients are tracked in a session database that stores the certificate details and is able to provide complete session logging for security and debugging purposes. The database is also used by the server to enforce certificate time limits. Other resource limits can be enforced e.g. CPU time, disk usage and network bandwidth usage.

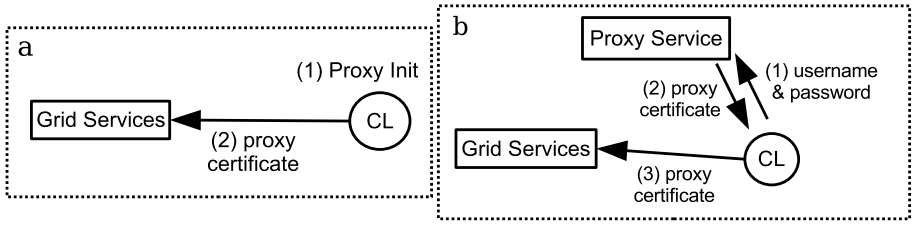


Fig. 2. Two methods of authentication: a) a temporary (proxy) certificate is created on the client machine and presented to Grid services for authentication. b) A proxy service is contacted using a username and password, and a temporary certificate is returned to the client machine. This temporary certificate can be presented to Grid services for authentication as usual

3 Building a Science Gateway

Increased computer security is always a compromise with ease-of-use. The agreement on reasonable levels of security is generally achieved by balancing these opposing requirements. In conventional use of computer centers, there are two main roles: the User and the Administrator. In contrast, when designing and building a science gateway, there are three roles:

- User: this is the person who uses the Gateway, and may be a skilled professional scientist, a schoolchild or anyone with intermediate expertise. This person may be an anonymous user, weakly authenticated or may be a strongly authenticated "power user". It is important to make sure that the former cannot run arbitrary code on the center's computers; and it is just as important not to overly restrict those who have gone to the effort of obtaining a high level of certification.
- Administrator: this person decides on certificate policies, audits code (or trusts the Gateway Developer, see below), and deploys the relevant services so they are accessible and reliable. Furthermore, the Administrator makes sure the Gateway resides in a *sandbox*, meaning that even if the Gateway Developer supplies insecure software, then the damage is limited.
- Gateway Developer: this person (or group of people) has the scientific domain knowledge (what is being computed and why), and also the programming skill to connect domain-specific code into the execution sandbox, and to decide appropriate levels of resource consumption. The Gateway Developer has the responsibility for limiting what the Users can do, and in particular for ensuring that anonymous or HotGrid Users cannot run arbitrary code.

The Administrator can build a sandbox for the gateway by keeping it within a *restricted environment* using the well-known method of partitioning a so-called `chroot` jail, meaning that only files within the sandbox can be read, written, or executed. The Developer, in turn, should ensure that clients with weak authentication cannot run shell commands that are not explicitly allowed for the

purposes of the science domain. The examples in section 4 illustrate these points in the context of real examples.

3.1 Accounting

We have described how a user with weak authentication can be allowed to use the power of a computer center, but without being given the power to run a command that may compromise system integrity. However, the user needs to be restricted further in the sense of resource usage. We do not want a weakly authenticated HotGrid user submitting a thousand jobs to a queue and block the queue with running jobs for weeks.

We envisage HotGrid users obtaining a certificate that has a near revocation date, perhaps one or two days in the future. This will prevent any use of the gateway after that time. Indeed, the user will be unable to access any information about sessions that were started with that certificate.

When a HotGrid certificate is presented, it will be mapped to a generic account name that for all HotGrid users of a particular gateway. These accounts will have an associated responsible "power user", who would generally be the gateway developer. The usage for these accounts will thus be an aggregate of all HotGrid usage.

Usage could also be restricted in a very application-specific fashion. As an example, an astronomical image mosaicker would not allow HotGrid users to create a mosaic larger than a given size. An HEP Monte Carlo event simulator would not allow more than a certain number of events to be generated.

4 Examples

4.1 Astronomical Image Processing

The objective here is a mosaic of astronomical images[14]; an example is in prototype on the Caltech TeraGrid, using the Atlasmaker[15] software. The user provides a string that is the name of an astronomical object or coordinates in the sky, that will be the center of the requested image, together with an angular size of the request. These parameters can be provided by web form or by client API, which sends the request via XML-RPC to the gateway. The user receives by immediate return a monitoring URL that contains the random sessionID. The user can then probe the progress of the of the job through this URL and collect results when the job is finished.

4.2 Monte-Carlo Processing and HEP

Another prototype is a science gateway that allows anonymous and HotGrid users to submit Monte-Carlo jobs that simulate proton-proton collisions in the CERN CMS (Compact Muon Solenoid) detector. Typically, a particular event topology, and a statistically significant number of events are required. The Monte Carlo is a standard piece of the CMS experiment's software stack, and ideally

suited to encapsulation as a gateway service. A form would be completed specifying the desired Monte Carlo generation parameters and number of events. The gateway would take care of parcelling the request into units that could be individually and independently executed on a cluster of computers running e.g. Condor[17], and aggregating the results datasets into a complete file which could then be downloaded by the user.

4.3 Grid Enabled Analysis of HEP Events

The ROOT software is widely used in HEP for the analysis of particle physics data. Akin to tools like Mathematica and Matlab, ROOT offers a rich suite of integrated tools for histogramming, fitting, rendering, annotating and otherwise processing scientific data. While ROOT is commonly run on a user's desktop or laptop, a ROOT server and slave system is available, called PROOF, that allows the event processing to be carried out in parallel on a set of worker computers that are coordinated by a master computer.

The PROOF system is an ideal configuration for an implementation of a HotGrid-based service along the lines described in this paper. A ROOT user, running on a resource-limited desktop system, and thus unable to analyse a sufficient number of events to achieve a statistically significant result, obtains a HotGrid certificate. With the certificate, she authenticates with a Clarens TeraGrid service offering the PROOF system, which then allows her to bring to bear the full power of a PROOF cluster of one hundred TeraGrid CPUs. Moreover, she never needs to leave the ROOT environment: all the dataset I/O and processing are handled in a transparent way, and the impression is simply that her desktop system has suddenly increased in power by at least two orders of magnitude.

4.4 VOStatistics and IRAF

The *VOStatistics* gateway is also in prototype, based on a web service framework developed previously[9]. It provides the client with access to a remote session of the "R" software, which provides powerful statistics and data mining capability. Similarly, the astronomical data processing package IRAF[11] is being made available by the US National Virtual Observatory[16] as a remote service package.

In each of these cases, the user enters a script that is executed by the server, perhaps processing thousands of files. There is, however, a potential danger, because these powerful environments all offer a shell escape command. Therefore we must be careful that we are not providing the ability to execute arbitrary shell commands – by means of restricted shells on the server, as discussed above.

The utility of remote execution is that the remote machine can be much more powerful than the user's workstation, perhaps because it holds a large archive or runs on a multi-processor machine. However, effectively utilizing R or IRAF on a distributed-memory cluster is still a research effort.

References

1. G. Aloisio, M. Cafaro, C. Kesselman, R. Williams: Web access to supercomputing, Computational Science and Engineering 3 (6): 66-72 (2001)
2. G. Aloisio G, M. Cafaro, P. Falabella, R. Williams Grid computing on the web using the globus toolkit, Lecture Notes in Computer Science 1823: 32-40 (2000)
3. Bunn J. and Newman H. Data Intensive Grids for High Energy Physics, in Grid Computing: Making the Global Infrastructure a Reality, edited by Fran Berman, Geoffrey Fox and Tony Hey, March 2003 by Wiley.
4. Conrad D. Steenberg, Eric Aslakson, Julian J. Bunn, Harvey B. Newman, Michael Thomas, Ashiq Anjum, Asif J. Muhammad Web Services and Peer to Peer Networking in a Grid Environment Proceedings of the 8th International Conference on Advanced Technology and Particle Physics, 2004
5. Arshad Ali, Ashiq Anjum, Tahir Azim, Michael Thomas, Conrad Steenberg, Harvey Newman, Julian Bunn, Rizwan Haider, Waqas ur Rehman JClarens: A Java Based Interactive Physics Analysis Environment for Data Intensive Applications Presented at the 2004 IEEE International Conference on Web Services (ICWS 2004), San Diego, July 2004
6. Julian J. Bunn, Harvey B.Newman, Michael Thomas, Conrad Steenberg, Arshad Ali and Ashiq Anjum. Grid Enabled Data Analysis on Handheld Devices Presented at International Networking and Communications Conference 2004(INCC,2004),Lahore,Pakistan.
7. Julian J. Bunn, Harvey B.Newman, Michael Thomas, Conrad Steenberg, Arshad Ali and Ashiq Anjum. Investigating the Role of Handheld devices in the accomplishment of Interactive Grid-Enabled Analysis Environment Proceedings of GCC2003 and Springer LNCS.
8. The ROOT Object Oriented Analysis Framework, `http://root.cern.ch`
9. G. J. Babu, S. G. Djorovski E. Feigelson, , M. J. Graham, A. Mahabal: Statistical Methodology for the National Virtual Observatory, `http://vostat.org`
10. The Interactive Data Language, `http://www.rsinc.com/idl/`
11. The Image Reduction and Analysis Facility, `http://iraf.noao.edu/`
12. Access to the TeraGrid, an overview, `http://www.teragrid.org/userinfo/guide_access.html`
13. Steenberg, C.D., et al.: The Clarens Web Services Architecture. Proc. Comp. in High Energy Physics, La Jolla, **MONT008**, 2003
14. R. D. Williams, Grids and the Virtual Observatory, in "Grid Computing: Making The Global Infrastructure a Reality" by Fran Berman, Anthony J.G. Hey, and Geoffrey Fox, Wiley, 2003, pp 837-858.
15. R. D. Williams, S. G. Djorgovski, M. T. Feldmann, J. C. Jacob: Atlasmaker: A Grid-based Implementation of the Hyperatlas http://arxiv.org/abs/astro-ph/0312196
16. The US National Virtual Observatory `http://www.us-vo.org`, also the International Virtual Observatory Alliance `http://www.ivoa.net`.
17. Thain, D. and Livny, M., Building Reliable Clients and Servers, in *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 2003.

Principles of Transactional Grid Deployment

Brian Coghlan, John Walsh, Geoff Quigley, David O'Callaghan,
Stephen Childs, and Eamonn Kenny

Department of Computer Science, Trinity College, University of Dublin
{coghlan, john.walsh, geoff.quigley, david.ocallaghan,
stephen.childs, ekenny}@cs.tcd.ie

Abstract. This paper examines the availability of grid infrastructures, describes the principles of transactional grid deployment, outlines the design and implementation of a transactional deployment system for the Grid-Ireland national grid infrastructure based on these principles, and estimates the resulting availability.

1 Introduction

Consider for instance that a new release of some infrastructural grid software is incompatible with the previous release, as is often the case. Once a certain proportion of the sites in a grid infrastructure are no longer consistent with the new release then the infrastructure as a whole can be considered inconsistent. Each grid infrastructure will have its own measures, but in all cases there is a threshold below which proper operation is no longer considered to exist. The infrastructure is no longer available. Thus availability is directly related to consistency. An inconsistent infrastructure is unavailable. As yet the authors are not aware of any prior availability estimates for existing grid infrastructures such as LCG[1], EGEE[2] or CrossGrid[3]. This is understandable in these early days of full deployment. Below we examine this subject, present some early estimates, propose a methodology, transactional deployment, that is intended to maximize the infrastructure availability, describe an implementation, and estimate the effect of the methodology on availability.

1.1 Consistency of a Single Site

Assume N identical sites are being evaluated in independent experiments, and that at time t, $C(t)$ are consistent (have upgraded) and $I(t)$ are inconsistent (have yet to upgrade). The probabilities of consistency and inconsistency are:

$$P_C(t) = C(t)/N \qquad P_I(t) = I(t)/N \tag{1}$$

where since $C(t) + I(t) = N$, then $P_C(t) + P_I(t) = 1.0$

The per-site upgrade rate is the number of upgrades per unit time compared with the number of sites that remain inconsistent:

$$U(t) = I(t)^{-1} dC(t)/dt \tag{2}$$

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 88–97, 2005.

This is the instantaneous upgrade rate $I(t)^{-1}\Delta C(t)/\Delta t$ for one site, i.e. the proportion that upgrade per unit time. Hence the average time a site waits before it becomes consistent (the mean time to consistency MTTC) is:

$$\text{MTTC} = \int_0^\infty P_I(t)dt \tag{3}$$

In some well regulated cases, e.g. 24×7 Operations Centres, it might be that the upgrade rate is close to a constant, say λ. In these cases the probability of a site being inconsistent (not upgraded) at time t and the resulting MTTC are:

$$P_I(t) = e^{-\lambda t} \qquad \text{MTTC} = \int_0^\infty P_I(t)dt = \lambda^{-1} \tag{4}$$

Bear in mind that all the above is true only if the upgrade rate is a constant. However, few organizations can afford facilities like 24×7 Operations Centres, so the general case is that the per-site upgrade rate is not constant, but a fitful function that reflects working hours, holidays, sick leave, etc. Moreover, upgrades at different sites are not independent, since it is usual that sites benefit from the experiences of sites that upgrade before them, but let us ignore this complication.

1.2 Consistency of a Multi-site Infrastructure

If the infrastructure contains M types of sites, then the Law of Probability states that the probability of all sites being inconsistent (none upgrading) or consistent (all upgraded) for time t is:

$$P_{I\text{infra}}(t) = \prod_{m=1}^{M} P_{Im}(t) \qquad P_{C\text{infra}}(t) = \prod_{m=1}^{M} P_{Cm}(t) \tag{5}$$

First let us take the case where all sites must be consistent for the infrastructure to be considered consistent. Then the probability of an inconsistent multi-site infrastructure is the probability of at least one site being inconsistent, i.e. the probability of NOT(all upgraded):

$$P_{I\text{infra}}(t) = 1 - P_{C\text{infra}}(t) = 1 - \prod_{m=1}^{M} (1 - P_{Im}(t)) \tag{6}$$

The more sites, the greater the probability. Clearly it is easier to understand the situation if all M sites are identical, i.e. the infrastructure is homogeneous, as is the case for Grid-Ireland. The MTTC is:

$$\text{MTTC}_{\text{infra}} = \int_0^\infty P_{I\text{infra}}(t)dt \neq \lambda_{\text{infra}}^{-1} \tag{7}$$

Note λ_{infra} is certainly not a constant, and the more sites, the longer the MTTC.

Now let us take the case where not all sites must be consistent for the infrastructure to be considered consistent. This might be the case for those grids where

there are a set of core sites, or those where some inconsistency is acceptable. Here we can use the Binomial expansion:

$$\prod_{m=1}^{M} (P_{Im}(t) + P_{Cm}(t)) = 1.0 \tag{8}$$

where $P_{Cm}(t) = 1 - P_{Im}(t)$. For M identical sites, then $(P_I(t) + P_C(t))^M = 1.0$ where $P_C(t) = 1 - P_I(t)$. For example, for $M = 4$:

$$P_I^4(t) + 4P_I^3(t)P_C(t) + 6P_I^2(t)P_C^2(t) + 4P_I(t)P_C^3(t) + P_C^4(t) = 1.0 \tag{9}$$

The first term represents the probability that no site has upgraded by time t, the second that any 1 has upgraded, etc. The probability of an inconsistent infrastructure representing ≥3 inconsistent sites, and the resulting MTTC, are:

$$P_{I\text{infra}}(t) = P_I^4(t) + 4P_I^3(t)P_C(t) = 4P_I^3(t) + 3P_I^4(t)$$
$$\text{MTTC}_{\text{infra}} = \int_0^\infty P_{I\text{infra}}(t)dt \neq \lambda_{\text{infra}}^{-1} \tag{10}$$

i.e. again, λ_{infra} is not a constant.

Thus if one knows the deployment behaviour of each site it is possible to calculate the MTTC of the infrastructure, especially if one uses symbolic algebra tools such as Maple or Mathematica.

1.3 Mean Time Between Releases

The interval between releases MTBR is quite independent of the MTTC. The MTTC is a deployment delay determined by the behaviour of the deployers, whilst the MTBR is dependent upon the behaviour of developers. Otherwise similar considerations apply.

1.4 Availability of a Multi-site Infrastructure

The availability of the infrastructure is given by the following equation:

$$A_{infra} = \text{MTBR}_{infra}/(\text{MTBR}_{infra} + \text{MTTC}_{infra}) \tag{11}$$

1.5 Estimates from Testbed Experience

Figure 1 shows derived histograms of the release intervals and deployment delays for the CrossGrid production and development testbeds. The use of this data is open to debate. Firstly one could argue for compensation for 8 hours of sleep per day. Secondly they are derived from email arrival times, not from actual event times. Steps have now been taken by the CrossGrid deployment team in FZK to log these events, so in future much more accurate data will be accessible.

The release intervals greater than 30,000 seconds (approximately 21 days) can be considered as outliers. Two intervals relate to the month of August, a

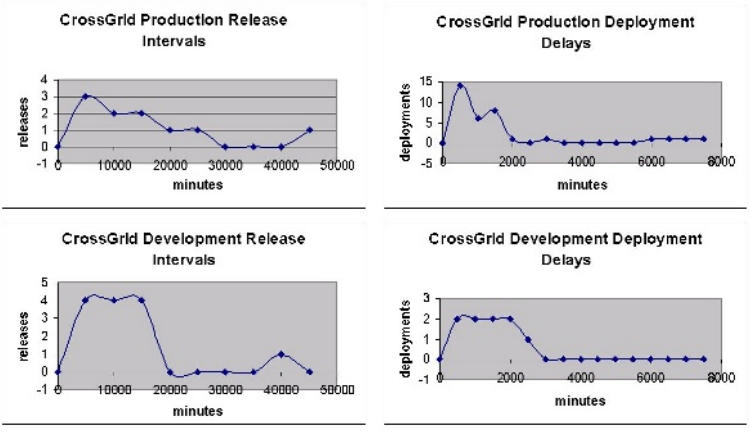


Fig. 1. CrossGrid testbed release intervals and deployment delays

Table 1. Observations for the CrossGrid production testbed

Observed MTBR	163 hours
Observed MTTC	11.5 hours
Observed availability	93.4%

Table 2. Observations for the CrossGrid development testbed

Observed MTBR	120 hours
Observed MTTC	18 hours
Observed availability	86.9%

traditional holiday month in Europe. Similarly the deployment delays greater than 3,000 seconds (approximately 2 days) might be considered outliers.

If we ignore the outliers, then from these figures we may derive a notional observed MTBR, MTTC and availability for the CrossGrid production and development testbeds, assuming consistency after the average deployment delay. Thus the observed infrastructure availabilities are as shown in Tables 1 and 2.

2 Principles

2.1 Two-Phase Transactionality

Maximizing availability means maximizing the proportion of time that the infrastructure is entirely consistent. This requires either the the time between releases

MTBR to be maximized or the MTTC to be minimized. The MTBR is beyond the control of those who manage the infrastructure. On the other hand, if the MTTC can be minimized to a single, short action across the entire infrastructure then the availability will indeed be maximized.

However, an upgrade to a new release may or may not be a short operation. To enable the upgrade to become a short event the upgrade process must be split into a variable-duration prepare phase and a short-duration upgrade phase, that is, a two-phase commit. Later in this paper we will describe how this can be achieved.

If the entire infrastructure is to be consistent after the upgrade, then the two-phase commit must succeed at all sites. Even if it fails at just one site, the upgrade must be aborted. Of course this may be done in a variety of ways, but from the infrastructure managers' viewpoint the most ideal scenario would be that if the upgrade is aborted the infrastructure should be in the same state as it was before the upgrade was attempted, that is, the upgrade process should appear to be an atomic action that either succeeds or fails.

Very few upgrades will comprise single actions. Most will be composed from multiple subactions. For such an upgrade to appear as an atomic action requires that it exhibits transactional behaviour, that all subactions succeed or all fail, so that the infrastructure is never left in an undefined state.

Thus we can see that to maximize availability requires that an upgrade be implemented as a two-phase transaction.

2.2 Preservation of ACID Properties

Since Gray[4, 5], transactions have been widely assumed to preserve so-called ACID properties. Let us consider each of these:

(a) *Atomicity:* if a transaction contains multiple sub-actions, these are treated as a single unit. Either all sites upgrade or none.
(b) *Consistency:* a transaction never leaves the infrastructure in an undefined state. It is enforced by a single distributed commit action in the case of success, or a distributed rollback in the case of an error.
(c) *Isolation:* transactions are separated from each other until they're finished. We only permit one transaction to take place at a time, bypassing this issue.
(d) *Durability:* once a transaction has been committed, that transaction will persist even in the case of an abnormal termination.

2.3 Homogeneous Core Infrastructure

A homogeneous core infrastructure is not strictly required for transactional deployment but is highly desirable. It greatly simplifies the understanding of the infrastructure availability, as can be seen from equations [7–10]. Logically, it allows a uniform control of the grid infrastructure that guarantees uniform responses to management actions. It also assures a degree of deterministic grid management. Furthermore it substantially reduces the complexity of the release packaging and process. Minimizing this complexity implies maximizing the uniformity of the deployed release, i.e. the core infrastructure should be homogeneous.

2.4 Centralized Control via Remote Management

Centralized control of the core grid infrastructure enables simpler operations management of the infrastructure. Remote systems management enables low-level sub-actions to be remotely invoked if a transaction requires it. It also enables remote recovery actions, e.g. reboot, to be applied in the case of deadlocks, livelocks, or hung hardware or software. Realistically, all infrastructure hardware should be remotely manageable to the BIOS level.

3 Transaction Algorithm

The transaction algorithm is:

```
ssh-add keys                                          //server startup
fork ssh tunnel(localhost:9000,repository:9000)
file read sitelist
XMLRPC server start
create lock                                           //transaction start
if(lock success) {
  set global_status (preparing)                       // Server prepare
  set release_dir(date, tag)
  cvs_checkout(date, tag)
  if (success) {                                      // prepare sites
    foreach(site in selected_sites) {
      sync RPMS, LCG-CVS, GI-CVS
      backup current GI,LCG links in release_dir
      backup profiles link
      create new GI,LCG links
      build site profile
    }
    if(failure) {                                     //rollback
      foreach(site in selected_sites) {
        restore link backup from release_dir
      }
      if( rollback_failure )
        set status rollback_error
    } else {                                          // commit
      foreach(site in selected_sites) {
        create new profiles link
} } } }
delete lock                                           //transaction end
```

4 Implementation

4.1 Architecture

Logically, the implementation divides into three segments. The first is the repository server, which hosts both the software to be deployed and the Transactional

Deployment Service (TDS) logic. Secondly, there is the user interface, which has been implemented as a PHP page residing on an Apache web server. Finally there are the install servers at the sites that we are deploying to. These servers hold configuration data and a local copy of software (RPMs) that are used by LCFGng to maintain the configuration of the client nodes at the site. It is the state of these managed nodes that we are trying to keep consistent.

4.2 Communication

The TDS establishes connections to the web server and the various install servers using SSH. The TDS is run within ssh-agent and it loads the keys it needs at start-up. Firewall rules on the repository server, where the TDS resides, and the web server, where the user interface resides, prevent unwanted connections. Susceptibility to address spoofing attacks is minimized by only allowing the TDS to create network connections.

The use of ssh-agent is very powerful, as it allows the TDS software to securely initiate multiple outbound connections as and when they are needed without any need for the operator to enter passwords. This ability was a key enabler in allowing all the control to come from the central server without having to push scripts out to each site. It also means that the TDS directly receives the return codes from the different commands on the remote sites. For extra security, when the remote sites are configured to have the TDS in their authorized keys files, the connections are restricted to only be accepted from the repository server, to disallow interactive login and to disallow port tunnelling (an important difference between the configuration of the ssh connection to the web server and to the remote sites!).

4.3 Transactional Deployment Server

The TDS consists of a Perl script and a small number of shell scripts. Thanks to the use of high level scripting languages the total amount of code is relatively small — approximately 1000 lines. Much of this code is in the main Perl script, which runs as an XMLRPC server[6, 7]. There are two reasons for the choice of XMLRPC over HTTP for the communications between the TDS and the user interface. Firstly, the XMLRPC protocol is not tied to a particular language, platform or implementation. There are numerous high quality and freely available implementations and they mostly work together without problems. Secondly, XMLRPC is simple and lightweight. Various other solutions were considered, such as SOAP communications with WS-Security for authenticating the communications at the TDS. However, WS-Security is still in development, and SOAP interoperability is in debate within the WS-I initiative.

XMLRPC exposes methods for preparing a site, committing changes and rolling back changes. Internally, the TDS uses a series of shell scripts to perform the actions associated with the various stages. There are a small number of these groups of actions to collect together as a single atomic prepare phase. Importantly, each action that takes place is checked for success and if any action fails then the whole group of actions returns an error. The error code returned can

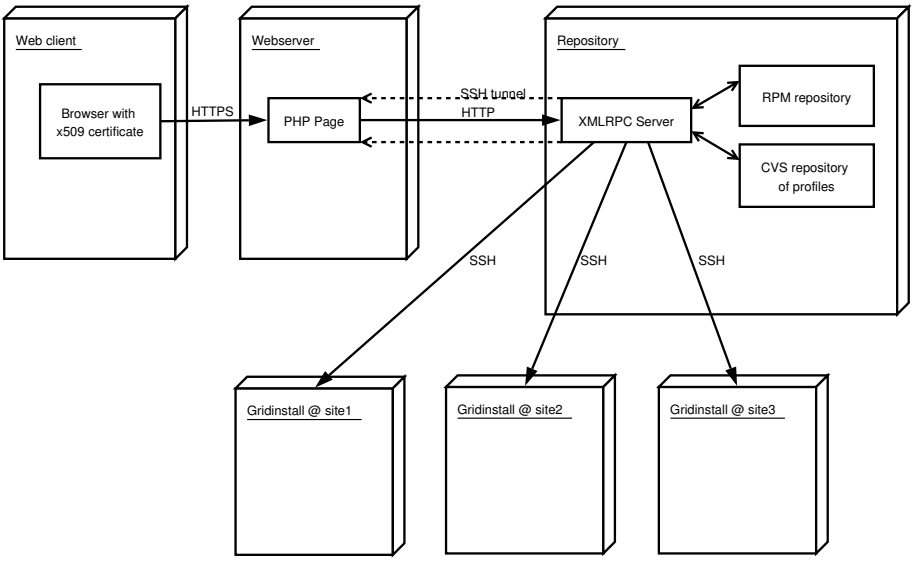


Fig. 2. Transactional Deployment System architecture

be used to determine exactly where the failure occurred. Only one softlink needs to be changed to roll back a remote site. The other changes that have taken place are not rolled back; this results in mild clutter on the remote file system, but does not cause significant problems. If the changes to the softlinks are not rolled back there is a possibility of incorrect configurations being passed through to the LCFG clients should an administrator attempt to manually build and deploy profiles. We intend to add extra functionality so multiple past transactions can be rolled back.

Locks are used to prevent more than one transaction being attempted at a time. Should the TDS exit mid-transaction, the lock file will be left in place. This file contains the process id of the TDS, so that the system administrator can confirm that the process has unexpectedly exited. There is no possibility of deadlock.

4.4 User Interface

The user interface is a simple page implemented in PHP and deployed on an Apache web server. Access to the page is only allowed via HTTPS and Apache is configured to only accept connections from browsers with particular certificates installed. The PHP code uses XMLRPC to communicate with the TDS to determine which sites are available for deployment and what their current state is. A drop-down list is populated with a list of the available version tags from the Grid-Ireland CVS repository. These tags are only set for known good releases of configuration settings. As previously stated, the TDS establishes an

SSH tunnel between the repository server and the web server, so the PHP code opens communications to a local port on the web server.

Users of the interface have five actions available. The simplest of these is to simply update the information displayed. The other actions are *prepare*, *commit*, *rollback* and a combined *prepare and commit* action. For the prepare actions, the user needs to select a release tag to deploy plus the sites to deploy to. Once the prepare action has been performed the control for selecting a release is disabled until either a commit or rollback has taken place. Commit and rollback do, however, accept selection of which sites to commit or rollback. Failure in the prepare phase will normally result in an automatic rollback. The normal usage of the system will be to select sites and a release tag, and then to use the combined prepare and commit action. This reduces the time between the prepare and commit phase, reducing the likelihood of problems appearing in this time, and ensuring that prepare and commit are carried out on the same list of sites.

5 Evaluation

It will be a while before statistically significant data is available for transactional deployment to a real grid infrastructure such as Grid-Ireland. The MTBR, i.e. the time between releases, is the same with or without transactional deployment. Let us assume the CrossGrid production testbed MTBR by way of example.

The transactional deployment delay is the sum of three consecutive delays: the commit time T_{commit}, the time T_{signal} for the LCFG install server to successfully signal its client nodes, and the time T_{update} it takes for a client node to update to the new release once it has recognized the server's signal. Our current estimates of these are: $T_{commit} = 20mSec$, $T_{signal} = 10minutes$ worst case, and $T_{update} = 7.5minutes$ worst case. The worst case value for T_{signal} results from the fact that client nodes check for the update signal on a 10 minute cycle. The worst case value for T_{update} is an average of 50 runs of a clean installation of a LCFG Computing Element, so it represents an extreme upper bound on representative physical machines. Therefore the worst-case MTTC is 17.5 minutes. The resulting worst-case infrastructure availability is shown in Table 3.

If the LCFG client nodes could be signalled to update immediately, i.e. $T_{signal} = 0$, then the availability would be increased 99.92%, and in fact this is likely to be substantially better for realistic release updates.

Table 3. Example MTBR, MTTC and availability for transactional deployment

Estimated MTBR	163 hours
Estimated MTTC	17.5 minutes
Estimated availability	99.82%

6 Conclusions

It is clear that the infrastructure has greatly enhanced availability with transactional deployment, improved from 87–93% to 99.8%, with the potential for 99.9% availability with a small amount of extra work.

The concept is certainly scalable to many tens of sites, but probably not hundreds, and almost certainly not at this time across national boundaries. How can this be accommodated? One obvious solution is to mandate compatibility across releases. The advantage is that a national commit is no longer necessary, although a site commit is still worthwhile to reduce the MTTC. The disadvantage is that it is very restrictive for developers. Another possibility is to avail of the evolving federated structures within international grids such as EGEE. Transactional deployment could be performed in concert across a federation by the relevant Regional Operating Centre [8], perhaps loosely synchronized across national boundaries. The mooted International Grid Organization could then act as a further level of loose synchronization between the Core Infrastructure Centres of each federation.

As far as the authors are aware, there has been no prior related work. Configuration tools such as GridWeaver[9] and PACMAN[10] manage software deployment on a single site rather than entire infrastructures.

Without doubt the most important benefit of the transactional deployment system not yet mentioned is the ease it brings to deployment. Transactional deployment allows for an automated totally-repeatable push-button upgrade process, with no possibility of operator error. This is a major bonus when employing inexperienced staff to maintain the grid infrastructure.

References

1. LCG: Large hadron collider computing grid project. http://lcg.web.cern.ch/LCG/ (2004)
2. EU: Enabling grids for e-science in europe (EGEE). http://www.eu-egee.org/ (2004)
3. EU: Crossgrid. http://www.crossgrid.org/ (2004)
4. Gray, J.N.: Notes on data base operating systems. In Bayer, R., Graham, R., Seegmuller, G., eds.: Operating Systems: An Advanced Course. Springer Verlag, New York, NY (1978) 393–481
5. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufman (1993)
6. Userland-Software: XML-RPC home page. http://www.xmlrpc.org (2004)
7. Ray, R.J., Kulchenko, P.: Programming Web Services with Perl. O'Reilly (2003)
8. EGEE: SA1: Grid operations. http://egee-sa1.web.cern.ch/egee-sa1 (2004)
9. University of Edinburgh and HP Laboratories: Gridweaver. http://www.gridweaver.org/ (2004)
10. Grid Research Integration Deployment and Support Center: PACMAN. http://www-unix.grids-center.org/r6/ecosystem/packaging/pacman.php (2004)

Experience with the International Testbed in the CrossGrid Project

J. Gomes[1], M. David[1], J. Martins[1], L. Bernardo[1], A. García[2], M. Hardt[2], H. Kornmayer[2], J. Marco[3], R. Marco[3], D. Rodríguez[3], I. Diaz[3], D. Cano[3], J. Salt[4], S. Gonzalez[4], J. Sánchez[4], F. Fassi[4], V. Lara[4], P. Nyczyk[5], P. Lason[5], A. Ozieblo[5], P. Wolniewicz[6], M. Bluj[7], K. Nawrocki[7], A. Padee[8,9], W. Wislicki[8], C. Fernández[10], J. Fontán[10], Y. Cotronis[11], E. Floros[11], G. Tsouloupas[12], W. Xing[12], M. Dikaiakos[12], J. Astalos[13], B. Coghlan[14], E. Heymann[15], M. Senar[15], C. Kanellopoulos[16], A. Ramos[17], and D. Groen[17]

[1] Laboratório de Instrumentacão e Física de Partículas, Lisbon, Portugal
[2] Forschungszentrum Karlsruhe GMBH, Germany
[3] Instituto de Física de Cantabria (CSIC-University of Cantabria), Santander, Spain
[4] Instituto de Física Corpuscular(CSIC-University of Valencia), Valencia, Spain
[5] Akademickie Centrum Komputerowe CYFRONET, Krakow, Poland
[6] Poznan Supercomputing and Networking Center, Poznan, Poland
[7] A. Soltan Institute for Nuclear Studies, Warsaw, Poland
[8] Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw,Poland
[9] Instytut Radioelektroniki PW, Warsaw, Poland
[10] CESGA, Centro de Supercomputacion de Galicia, Santiago de Compostela, Spain
[11] National Center for Scientific Research "Demokritos", National and Kapodistrian University of Athens, Dep. of Informatics and Telecommunications, Greece
[12] University of Cyprus, Cyprus
[13] Ustav Informatiky Slovenska Akademia Vied, Bratislava, Slovakia
[14] Trinity College Dublin, Ireland
[15] Universitat Autonoma de Barcelona, Spain
[16] Aristotle University of Thessaloniki, Greece
[17] Universiteit van Amsterdam, Netherlands

Abstract. The International Testbed of the CrossGrid Project has been in operation for the last three years, including 16 sites in 9 countries across Europe. The main achievements in installation and operation are described, and also the substantial experience gained on providing support to application and middleware developers in the project. Results are presented showing the availability of a realistic Grid framework to execute distributed interactive and parallel jobs.

1 Introduction

The European CrossGrid project [1] has developed new components for interactive compute and data intensive applications in a Grid [2] framework.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 98–110, 2005.

The main objective of the CrossGrid testbed was to provide the framework where the new Grid services and programming tools could be developed and tested, and interactive user-friendly applications executed.

One of the initial objectives was to assure full interoperability with the European DataGrid (EDG) project [3] middleware to profit from their results and experience, and achieve an extension of this basic Grid framework across eleven European countries. This coordination has made possible also the successful contribution of many of the CrossGrid partners to the EGEE initiative [4] to setup a production-level Grid infrastructure in Europe.

Another important objective was to support the execution of parallel applications using the MPI [5] protocol, running either inside a cluster (using MPICH-P4 [7]) or across different sites (using MPICH-G2 [6]), and of the different services and tools oriented to support interactivity.

This article summarizes the experience gained not only on the deployment and operation of the distributed computing resources, but also on the corresponding support provided to developers and users.

Further technical details can be obtained from the project technical deliverables, and are also described in the CrossGrid book [8].

2 Implementation of the CrossGrid International Testbed

The CrossGrid international distributed testbed [9] shares resources across sixteen European sites and this is itself one of the challenging points of the project. The sites range from relatively small computing facilities in universities to large computing centers, offering an ideal mixture to test the possibilities of the Grid framework. National research networks and the high-performance European network, Geant, assure the inter-connectivity between all sites. The network includes usually three ranges: the local campus (typically inside a University or Research Center, via Fast or Gigabit Ethernet), the connection via the national network provider (at speeds that range from 34 Mbits/s to 622 Mbits/s or even Gigabit) to the national backbone, and finally the link to the Geant network (155 Mbits/s to 2.5 Gbits/s). The figure 1 shows a map with the different testbed sites, including the major network links.

As indicated before, the basic Grid middleware was selected to guarantee interoperability with the large EDG project testbed.

At the most lower level, the basic job submission services, information systems, authentication, authorization and data transfer are provided by the Globus Toolkit [10]. These services are extended by a workload management system, replica location services, improved information system and Virtual Organization management system developed by the EDG project, incorporating also other middleware components from packages like Condor [11]. This software is currently distributed in a version known as LCG-2, assembled and supported by the LHC Computing Project at CERN [12].

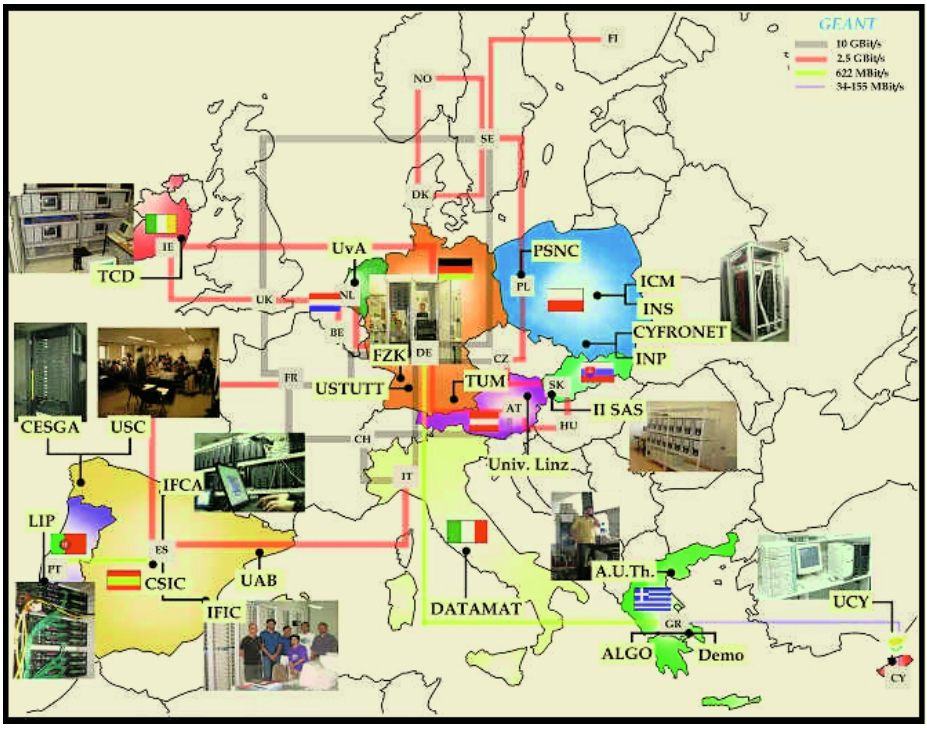


Fig. 1. The CrossGrid testbed sites

On top of these two basic layers, CrossGrid has developed its own set of services:

- User friendly Migrating Desktop (MD) to access the Grid environment, complemented by a Roaming Access Server (RAS) and integrating support for Interactivity.
- Improved workload management system (WMS) with MPI local and distributed support, and prioritization mechanism.
- Monitoring systems for infrastructure (JIMS), network traffic (SANTA-G), application execution (OCM-G), and resource usage prediction (GMDAT)
- Data access optimization (UNIDAL)

All these services are used in the programming tools developed within the project to help application teams to tune their software for the grid environment:

- Grid benchmarking tools (GRIDBENCH)
- Performance Prediction for application kernels (PPC)
- Performance Monitoring tool for distributed applications (GPM)
- MPI code debugging and verification tool (MARMOT)

Finally, the testbed provides the Grid environment where four parallel and interactive applications on different areas (BioMedicine, Flooding Crisis Management, Particle Physics, and Meteorology and Air Pollution) have been developed making use of these services and tools. The testbed also provides the framework for the test and execution of these data and compute intensive applications.

For all these purposes, the final version of the CrossGrid testbed includes two distinct setups. The "development" testbed, with limited resources in five different sites, supports deployment and test of new software, that once validated is deployed in a larger and more stable "production" testbed, where the applications are executed.

Grid testbeds provide two different types of services: global collective services, and computing and data services using resources distributed across the sites.

In the CrossGrid testbed each site includes the following computing and data storage resources:

- A Computing Element (CE) machine providing the interface between the grid and the local processing farm.
- An Storage Element (SE) machine providing the interface between the grid and local storage resources.
- A User Interface (UI) machine enabling the local users to access the testbed.
- A farm with at least two Worker Nodes (WN) or a dual CPU system.
- An installation server using the LCFGng [14] software.

The CrossGrid testbed includes a total of about 200 CPUs and a distributed storage capacity above 4 Terabytes.

The integration of these resources is done by the global collective services:

- Resource Broker (RB): the heart of the testbed workload management system. The RB receives job requests sent by users and finds computing resources suitable to run the jobs. The CrossGrid RB supports single and parallel jobs (inside a cluster via MPICH-P4 or distributed using MPICH-G2)
- Information Index (II): the root entry point for the MDS information tree that contains the resources information published by the CE and SE systems.
- MyProxy: repository of certificate proxies for long-lived jobs and portals.
- Virtual Organization server: repository of authorization information used by the testbed systems to generate the authorization databases. The CrossGrid VO server contains the list of all CrossGrid users authorized to use the testbed.
- Replica Location Services (RLS): a service that stores information about the location of physical files in the grid. The RB uses the RLS to make scheduling decisions based on the location of the files required by the jobs. The users can also use the RLS service through the Replica Manager (RM). Although this is an EDG service CrossGrid is developing components for its access optimization functionality.
- Mapcenter: an EDG tool enhanced by CrossGrid to monitor the connectivity of systems and services.
- Roaming Access Server (RAS): CrossGrid specific service that provides support for the CrossGrid Migrating Desktop (MD) and portal applications.

- Roaming Access Server Job Submission Service (RAS JSS): a component of the CrossGrid RAS server that performs the actual job submission.
- Grid Monitoring Data Analysis Tool (GMDAT): a service to monitor and predict the usage of grid resources and the network bandwidth between sites.
- Relational Grid Monitoring Architecture (R-GMA): a service to enable the publishing of monitoring information in distributed grid enabled relational database tables.

The global collective services are located mainly in four sites. Resulting from the collaboration between the Laboratório de Instrumentacão e Física de Partículas (LIP) in Lisbon and the Portuguese academic network (FCCN) on Grid technologies the main production central systems at the LIP/FCCN site are physically hosted at the FCCN Network Operations Center in Lisbon. The systems share a 100Mbits/s network connection to the multi-Gigabit Pan-European network backbone Geant. The FCCN collaboration allowed the the production RB to be installed in a Geant point of presence improving the network connectivity and resilience of this critical service. The less demanding central production services have been hosted at the LIP computer centre facilities also in Lisbon. Redundancy has been introduced into the production central services with the duplication of the most critical central services at the Instituto de Física de Cantabria (IFCA) site in Santander. This duplication allows the production testbed to continue to operate in case of a major failure at the Lisbon sites.

The main global services for the development testbed are located at the Forschungszentrum (FZK) in Karlsruhe.

Several other sites contributed to the collective services support effort by hosting services related with specific project developments for both the production and development testbeds.

Security services in the CrossGrid infrastructure rely on the Globus Security Infrastructure (GSI) using public key cryptography based on X.509 certificates. GSI provides the ability for secure communications over the Grid and for decentralized inter-organizational authentication enhanced by single sign-on features and delegation of credentials.

Ten national Certification Authorities are involved, six of them setup thanks to the CrossGrid project. All of them are now members of the European Grid Policy Management Authority (euGridPMA) [13], and serve many other current Grid projects across Europe.

3 Support Tools in the CrossGrid Testbed

The support within the project can be divided in three categories according to the target communities: “user support”, “developer support” and “site administrator support”. Site administrators are responsible for the installation, maintenance and daily operation of the testbed sites. They need mostly good documentation covering the software installation and configuration, and tools to help them to verify their sites configuration and behavior. Developers are special users that are deeply involved in writing software and as such need support not

only in the testbed usage but also in the software testing, packaging, integration and deployment. Developers also need access to specialized tools such as a central CVS [20] repository, bug tracking tools and packaging tools. End-users are interested in running the applications accessing to the testbed in a user friendly way and being isolated as much as possible from the software and infrastructure complexity.

To satisfy the needs of these different communities several support channels have been created, from mailing lists to a dedicated helpdesk. Relevant links for coordination and support are provided from the testbed workpackage (WP4) main web page (http://grid.ifca.unican.es/crossgrid/wp4).

3.1 Site Administrators Support

Support for site managers is available through the "Testbed Support web site" at LIP. It includes three sections dedicated to the production, development and validation activities, and provides information about the testbed infrastructure and also on installing and configuring sites. The end users can also found relevant information about the setup of the testbed and examples on how to use it.

Additionally the site also hosts the testbed verification and quality control web pages. These pages contain information about the test and validation procedures for testbed sites and software packages, a validation request web form and a database with information about the validation requests and corresponding reports is also available.

The complexity of the CrossGrid software releases requires specialized tools to make the deployment process lightweight, fast and reliable. The release process relies in the software packing using the RedHat Package Manager (RPM) format. The generate auto-built packages are made available through the GridPortal repository at FZK jointly with configuration profiles for automated installation using the LCFG [14] installation and configuration software. These profiles contain configuration information that is common to all sites and are complemented by the site specific information provided by the systems administrators at each site. To further automate the deployment process, the `cg-lcfg-go` [23] tool was developed.

This approach assures that all sites have the same package versions configured properly and contributes to reduce the deployment time.

Additionally, a site installation manual "Cluster Installation in CrossGrid" [27], created and maintained by UoA and available from this "Testbed Support" web site, extensively explains all steps required to setup a new site, from the preparation of a LCFGng installation server to the deployment of a whole cluster. The manual has proved to be an excellent source of information for the sites installation, upgrade and maintenance.

3.2 Developers Support

All CrossGrid partners contribute to the project development effort. In such a complex development environment clear guidelines are needed to coordinate the development process. Using the EDG [3] developer's manual as input FZK

has created and maintained the official development reference manual [17] for CrossGrid.

Additionally FZK supports central development services including a CVS [20] repository, bug tracker, webspace and autobuild.

The autobuild [21] tool was developed by EDG and adapted for CrossGrid use. The tool takes the source code from the repository and produces RPM packages suitable from deployment. This fully automated process is performed every four hours.

3.3 User Support

The CrossGrid HelpDesk system is the main generic tool of the User Support Team. The system is installed at the Instituto de Física Corpuscular (IFIC) in Valencia, and can be accessed in http://cg1.ific.uv.es/hlpdesk.

It is a web based system based on the OneOrZero software [26], version v1.2 RC2 Red Lava incorporating PHP [15], JavaScript and MySQL [16]. This product is designed to be fully customizable.

All kind of questions related to the Testbed (i.e. CrossGrid V.O., Testbed installation and upgrades, network, security, resource broker, etc.) can be posed by users creating classified tickets, that are answered by a team of specialized supporters.

A Knowledge Base contains the stored problems reported by users with their corresponding solutions.

Another important tool for user support is the CrossGrid Tutorial, that includes a section with exercises on the CrossGrid Testbed, intended for new users that would like to run their jobs or applications in the Grid framework.

This tutorial explains all the steps that a user must follow to be able to execute jobs in the testbed: from obtaining the digital certificate to running MPI jobs and using the Replica services. All the examples, including source code, are available from http://grid.ifca.unican.es/crossgrid/wp4/Tutorial/Examples.

4 Test and Validation

The CrossGrid testbed workpackage includes a task dedicated to the testbed verification and quality control. The task aims to ensure that the CrossGrid testbeds conforms to the quality requirements of a near production service. Two main areas of activity emerged from the task objectives:

- Software test and validation.
  - Validation of the software to be deployed in the testbed.
  - Validation of software documentation.
- Testbed test and validation.
  - Validation of sites after each new installation or upgrade.
  - Continuous monitoring of the testbed.

The software test and validation is the last step of the CrossGrid quality assurance process and is responsible for the execution of the software acceptance tests. The testbed validation is responsible for the verification of the CrossGrid sites and services.

4.1 Software Validation

The software test and validation procedure followed by CrossGrid is defined as a set of interactions between the "verification and quality control team", the "integration team" and the developers. The procedure covers the documentation, the installation and the software functionalities. The documentation provided by the developers is followed to install the software in systems dedicated or allocated for the validation. The software is then tested using unit tests to verify the correct behavior of the software. System tests with the software integrated with the remaining testbed components are then performed. These tests are performed first at local level and then between sites if applicable. Finally stress tests are performed exercising the components.

The procedure was refined and improved along the project. Two major changes were introduced in 2004, the possibility of bypassing the validation procedure for minor fixes and the use of the bugtracker to keep a record of the detected issues.

The problems detected during the validation are classified regarding their severity and priority. These two attributes define the urgency and impact of the problem and help to determine which issues must be addressed first. For each severity level described a guideline action is recommended.

The validation procedure has been applied with good results. Providing immediate feedback on detected problems results usually in prompt actions by the developers, the cycle helps improving the stability of the middleware.

While applying the test and validation procedure to the CrossGrid software packages produced until November of 2004 a total of 142 software issues and 34 documentation issues have been detected these issues are distributed as follows:

4.2 Testbed Sites Validation

One of the most important steps toward the assurance of the correct site deployment is the site validation. This is a procedure intended to validate testbed sites after each major change, namely when a new middleware release is deployed.

Table 1. Issues found during validation

Severity		Priority	
Medium	72	Medium	57
High	70	High	64
Critical	21	Immediate	38
Low	13	Low	17

The procedure starts with the installation and configuration of the site following the CrossGrid site deployment manual and the support web pages. The installation and configuration were initially performed with LCFG and later with LCFGng to reduce the number of possible installation and configuration problems. The software release is downloaded from the CrossGrid repository at FZK where the common LCFG profiles are also available. The installation and configuration are responsibility of the site administrator. Once the site is deployed the site administrator contacts the testbed administrators and provides the site details.

The site is added to the CrossGrid "mapcenter" grid monitoring service and the connectivity of the systems and services is tested. If all services are reachable, the site CE and SE are added to the "Host Check" verification tool, a web enabled host verification tool that is capable of providing several installation, configuration and operation diagnostics. Once the nodes are added the site administrator can see by himself the list of problems detected and take appropriate measures to correct them. In this phase a strong interaction between the site administrators, the testbed administrators and the quality control team is required to help site administrators quickly solve any problems that are detected and bring the site up to the required quality level. The interaction is also important to reduce the testbed deployment time and understand problems and situations that may have not been identified before.

Once all the problems that were detected have been corrected, stress tests are performed on the site by submitting a large number of jobs and performing a large number of file transfers. These tests cover job submission through Globus and through the Resource Broker.

When the site is found to be stable, it is added to the list of official testbed sites and authorized to join the top MDS information index where all sites are registered.

The procedure has been successfully applied helping to locate many problems that would pass unnoticed until the sites would be actually used by the users possibly creating major testbed disturbances.

4.3 Testbed Monitoring

Sites must be continuously monitored both during the acceptance process and during operation. The aim of monitoring is to detect problems that can disturb the testbed and therefore contribute to instability. Monitoring also allows gathering statistics about the testbed that are useful to spot problems and evaluate its quality and usage.

The following tools have been developed at LIP and are used to perform testbed and site monitoring:

- Mapcenter: developed by DataGrid, it has been improved by CrossGrid with the support for usage statistics and testbed services uptime.
- CE usage statistics: Collects and presents information about the jobs submitted to Gatekeepers. Data is obtained from the gatekeepers log file and processed to produce statistics about the job submission errors.

- RB usage statistics: Collects information about the jobs submitted through a Resource Broker. The statistics are collected from the RB logging and bookkeeping database and are processed to produce usage graphics.
- Host Check: Host check is the main tool used in the site validation process. It is designed to verify the installation and configuration of Computing Elements and Storage Elements. Host Check is capable of detecting most of the known configuration and installation problems.
- CRL verification tool: The CRL verification tool performs CRL expiration checks by downloading the latest CRL's from the CA's web sites and verifying their expiration time.
- Site uptime: Collects and processes the Mapcenter alarms data to produce an historical view of the systems and services uptime.

5 Results

The experience with the CrossGrid testbed has shown that is feasible to use grid technologies to support interactive and parallel applications. However this class of applications requires careful testbed planning, deployment and operation thus infrastructure monitoring is a key factor for success.

Deploying and maintaining a grid site is a complex task that requires a considerable degree of knowledge covering many aspects of the operating system, grid middleware, security and networking. Experience has shown that manual configuration although possible can be difficult and highly susceptible to errors. Furthermore human resources with the necessary knowledge and practice to perform these operations correctly are frequently not available at all sites. For grid technologies to become widely available it is necessary to simplify the site management tasks.

Following the approach introduced by DataGrid the site management in CrossGrid was initially performed with the LCFG installation and management software. At a later stage LCFG was replaced by its successor LCFGng that become the official installation method until the end of the project. In spite of the problems encountered, the CrossGrid testbed has been successfully managed with LCFG. To help the site administrators in the deployment and daily maintenance of their systems a LCFG cluster installation and management manual [27] was written by UoA and common configuration profiles for each new release have been developed by FZK greatly contributing to a reduction in the deployment and upgrade times and enabling the maintenance of a homogeneous infrastructure.

The CrossGrid testbed supports parallel applications using both MPI restricted to clusters and MPI across clusters. Supporting MPI applications requires careful testbed configuration. Parallel applications that use MPI across multiple machines have an increased probability of failure in comparison with non-parallel applications.

The approach selected was to extend the DataGrid workload management system to support the submission of MPI jobs using two methods:

MPI inside clusters: Job submission from the resource broker to the best possible cluster according to the job specified requirements and using globus GRAM. The job runs inside a cluster using MPICH-P4.

MPI across clusters: Job submission from the resource broker to the best possible clusters using globus GRAM. Each application instance is started as a separate globus sub job, and can be run across different clusters. The communication mechanism between the sub jobs is provided by MPICH-G2, an MPICH implementation that uses the globus communication mechanisms (globus IO).

The last versions of the modified resource broker have shown remarkable MPICH-P4 job submission reliability to which the careful testbed monitoring and configuration has also greatly contributed.

The support for MPICH-G2 in the resource broker become available in 2004. The experience showed several new problems. The most common problem is caused by sub jobs that stay queued at a site while the remaining sub jobs enter into execution at other sites. Interactivity also introduced similar problems, as interactive applications need to be started immediately. This basic requirement collides with the nature of the load balancing systems used today in most clusters, such as PBS. These systems are designed to support the requirements of batch applications, where waiting in a queue does not constitute a problem. The interactivity issues are extremely similar to the ones from which MPICH-G2 suffers. A patch for the PBS job manager as also been developed within CrossGrid to specify that a job must enter immediately into execution, if the job remains queued it will be killed giving a change for the resource broker to restart it at some other location, or returning the control to the user.

Table 2. Testbed metrics

Month	Sites	Users			Jobs			Uptime
		Tot	Act	Ratio	Tot	Ok	Ratio	
2003 Aug	16	79	28	0.35	11424	11059	0.97	90%
2003 Sep	16	79	39	0.49	9278	8781	0.95	85%
2003 Oct	16	81	32	0.40	8919	8772	0.98	82%
2003 Nov	16	83	32	0.39	3118	1950	0.63	56%
2003 Dec	16	87	29	0.33	1627	1565	0.96	77%
2004 Jan	16	92	46	0.50	16882	16526	0.98	76%
2004 Feb	15	96	40	0.42	17471	17394	0.99	92%
2004 Mar	15	100	55	0.55	39674	39357	0.99	
2004 Apr	15	101	40	0.40	18620	18501	0.99	94%
2004 May	15	104	43	0.41	18648	18307	0.98	87%
2004 Jun	15	105	42	0.40	25687	24403	0.95	94%
2004 Jul	15	109	56	0.51	27326	27248	0.99	98%
2004 Aug	15	111	31	0.28	6148	6125	0.99	96%
2004 Sep	15	114	44	0.39	15381	14610	0.95	95%
2004 Oct	16	119	57	0.48	23279	23235	0.99	97%

A more sophisticated method to support interactive applications based on the Condor glide-in feature is also being introduced in the resource broker. This method aims to bypass the local load balancing system when submitting interactive applications. This method ensures that interactive jobs are started immediately and faster.

All these situations require careful resource monitoring. The host check tool was improved with tests that include job submissions using MPICH thus detecting the sites with problems. Tools developed at FZK also allowed the detection of SSH issues inside the clusters and the monitoring of relevant queue parameters.

The number of jobs submitted and jobs successfully executed since last year shows a clear improvement during the last months corresponding to the deployment of more stable middleware releases, including better support for MPI jobs in the workload management software.

The table 2 shows the evolution of the testbed quality indicators for the production testbed since August of 2003. The Table shows for each month the number of testbed sites, registered users, active users, jobs submitted, jobs sucessfull, and the average testbed uptime obtained from the Mapcenter monitoring tool. It can be observed the growth of the number of users, the improvement of the job submission ratio and of the average testbed uptime. The uptime values for March of 2004 were not collected due to the upgrade of the monitoring software.

6 Conclusions

The installation and operation of the CrossGrid International Testbed has been described. More than 200 CPUs and 4 Terabytes of storage are available for application execution.

Despite the intrinsic difficulty of organizing a real distributed Grid framework across 16 different sites, in 9 different countries, and thanks to the implementation of strong quality assurance mechanisms and support tools, the statistics show that the testbed is functional and allows the development and execution of interactive jobs and parallel applications.

Further effort along the next months will be devoted to improve on those areas more critical for the execution of interactive jobs, like prioritisation mechanisms, role of the load balancing systems, and optimized data transfer.

Acknowledgements

This work has been supported by the European project CrossGrid (IST-2001-32243) and by the Polish Committee for Scientific Research through the grants KBN/115/SPB/5.PRUE/DZ206 and KBN/115/SPB/5.PRUE/DZ208.

The authors would like to thank their colleagues in EDG and EGEE projects for their nice collaboration, and also to the CrossGrid reviewers that have pushed for the improvement of quality assurance mechanisms and support tools.

References

1. The European Project CrossGrid: http://www.eu-crossgrid.org
2. I. Foster and C. Kesselman (eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999.
3. The European Project DataGrid: http://www.edg.org
4. The European Project EGEE (Enabling Grids for e-Science): http://www.eu-egee.org
5. Message Passing Interface Forum, MPI: A Message Passing Interface Standard, June 1995. http://www.mpi-forum.org
6. A grid enabled implementation of the MPI standard: http://www3.niu.edu/mpi/
7. An implementation of the MPI standard for applications running inside clusters: http://www-unix.mcs.anl.gov/mpi/mpich
8. The CrossGrid Project Book, in press, Springer LNCS Series.
9. J. Gomes et al., "First Prototype of the CrossGrid Testbed", Proc. AcrossGrids Conf., Santiago, February 2003; ISBN 3-540-21048-2 Lecture Notes in Computer Science 2970: 67-77, 2004.
10. The Globus Project: http://www.globus.org
11. Condor Project Homepage: http://www.cs.wisc.edu/condor
12. The LHC Computing Grid Project: http://lcg.web.cern.ch
13. J.Astalos et al., "International Grid CA Interworking". Submitted to EGC'05, http://www.eugridpma.org
14. The Local Configuration software: http://www.lcfg.org
15. The Hypertext Preprocessing scripting language: http://www.php.net
16. The MySQL relational database: http://www.mysql.org
17. The CrossGrid Developers' Guide: http://gridportal.fzk.de/websites/crossgrid/iteam/devguide/devguide-html
18. The CrossGrid central software repository and development server, GridPortal: http://gridportal.fzk.de
19. The Savannah software: http://gna.org/projects/savane
20. CVS Documentation: http://www.cvshome.org
21. Autobuild entry page: http://savannah.fzk.de/autobuild/i386-rh7.3-gcc3.2.2
22. Autobuilt RPMs: http://savannah.fzk.de/distribution/crossgrid/autobuilt
23. cg-lcfg-go: http://cvs.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp4/config/cg-wp4-lcfg-server-files/cg-lcfg-go
24. Jorge Gomes et al., "CrossGrid Deliverable D4.1 Appendix D"" (http://www.lip.pt/computing/projects/crossgrid/doc/deliverables/TestProcedure.pdf)
25. CVS Documentation: http://www.loria.fr/~molli/cvs/doc/cvstoc.html
26. OneOrZero software: http://helpdesk.oneorzero.com
27. The CrossGrid cluster installation manual: http://cgi.di.uoa.gr/ xgrid/archive.htm

eNANOS Grid Resource Broker

Ivan Rodero, Julita Corbalán, Rosa M. Badia, and Jesús Labarta

CEPBA-IBM Research Institute,
Technical University of Catalonia (UPC), Spain
{irodero, juli, rosab, jesus}@ac.upc.es

Abstract. Grid computing has been presented as a way of sharing geographically and organizationally distributed resources and of performing successfully distributed computation. To achieve these goals a software layer is necessary to interact with grid environments. Therefore, not only a middleware and its services are needed, but it is also necessary to offer resource management services to hide the underlying complexity of the Grid resources to Grid users. In this paper, we present the design and implementation of an OGSI-compliant Grid resource broker compatible with both GT2 and GT3. It focuses in resource discovery and management, and dynamic policies management for job scheduling and resource selection. The presented resource broker is designed in an extensible and modular way using standard protocols and schemas to become compatible with new middleware versions. We also present experimental results to demonstrate the resource broker behavior.

1 Introduction

Grid computing [1] has emerged in recent years as a way of sharing heterogeneous resources distributed over local or wide area networks and geographically and organizationally dispersed. Grid computing builds on the concept of distributed computing, and software provides a way to divide up tasks so they are processed in parallel. In that context Grid computing is a good framework for solving large-scale problems such as bioinformatics, physics, engineering or life sciences problems.

In order to provide the necessary infrastructure for the Grid several projects have been developed such as *Globus Toolkit* [2], Condor [3] or Unicore [4]. In particular, Globus Toolkit is being implanted in several projects, with the aim of providing a generic solution for a Grid infrastructure.

In addition to the infrastructure basic services to give support to paradigms like *Resource Management* [6] are also required. The resource management in Grid environments is different from the one used in cluster computing. Recently, many efforts have been devoted to HPC, especially in job scheduling policies and resource usage maximization. Globus Toolkit provides some useful services including Grid Security Infrastructure (GSI) [7], Grid Resource Allocation and Management (GRAM) [8], Data Management Services (e.g. gridFTP) [9], and Information Services, Monitoring and Discovery System (MDS) [10].

Discovering and selecting suitable resources for applications in Grid environments is still an open problem. Thus, when a user wants to interact with a Grid, all processes

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 111 – 121, 2005.

related to resource management decisions should be handled manually. But these tasks are too difficult for a user and it appears to be a good idea to take a *Resource Broker* or a meta-scheduler to perform these basic functions. Additionally, no resource broker is included in top of the Globus Toolkit.

The main motivations for developing this resource broker are developing a resource broker compatible with emerging technologies such as Globus Toolkit 3 and accomplish the requirements of eNANOS project. When we started this project, no resource broker had been developed on top of Globus Toolkit 3.

In this paper, we present the design and implementation of an OGSI-Compliant resource broker developed as a Grid Service. The main objective is to expose the broker architecture and its characteristics not an evaluation. Our resource broker is compatible with both Globus Toolkit 2 and Globus Toolkit 3 services, and implements flexible mechanisms to become compatible with next Globus versions. It is centered in resource management and focuses on *dynamic policy management*. This resource broker is responsible for the Resource Discovery and Monitoring, Resource Selection, Job Submission and Job Monitoring; and implements policy management mechanisms from user side. It supports different policy classes including scheduling policies, resource selection policies and complex policies (called meta-policies). It uses a XML based language to specify user multi-criteria. It also provides a set of Grid Services interfaces and Java API for various clients, e.g. user applications, command-line clients or grid portals. Furthermore, we expose the main problems encountered in developing a resource broker on top of Globus Toolkit 3.

The rest of this paper is organized as follows. Section 2 overviews previous research on resource brokering and scheduling. Section 3 discuss the system design and implementation details of our Grid resource broker. Section 4 describes experimental results and section 5 concludes the paper and presents future work.

2 Related Work

At the moment there are many projects related to Grid since it is an important research issue for the international community. Some projects, such as AppLes [11], Nimrod/G [12], Condor-G [13], EZ-Grid [14], GridLab Resource Management System (GRMS) [15] or GridWay [16], have been working on brokering systems. These projects are developed on top of GT2 but other initiatives have been presented, for instance a Grid Broker Service [17] in terms of OGSA running on GT3.

Our Grid resource broker differs from previous existing brokerage systems in the following aspects: First, this general-purpose resource broker is compatible with GT2 and GT3 services, it implies that a uniform internal representation of objects and data involved in any task of resource management is needed; secondly, the proposed resource broker provides dynamic policy management which combined with user multi-criteria requirements allows us to advanced users a large capacity of decision. This user multi-criteria file is a XML document; it can be used in policies evaluation and is composed of requirements and recommendations. A requirement (hard attribute) is a restriction for resource filtering and a recommendation (soft attribute), "with its priority," can be used to provide a resource ranking for policies evaluation. Finally, since our resource broker is implemented as a grid service, we can have several broker instances to construct more scalable systems.

3 System Design and Implementation

3.1 Overall Architecture

This subsection presents the overall architecture of the proposed Grid resource broker. As shown in Fig. 1, the broker consists of five principal modules, a queuing system and data system for persistency. Moreover, the system is composed of Globus Toolkit services and an API to access the broker services.

Resource Discovery uses both GT2 MDS (GRIS/GIIS servers) and GT3 Information Services (based in Web Services). It uses a uniform representation of resource servers and resources based on GLUE schema.

Resource Selection performs dynamic selection of best resources from job specifications, user criteria, resource information and policies evaluation. All decisions related to resources are made from the local data obtained in resource discovery and monitoring processes.

Resource Monitoring gathers information about resources and stores it as local information which is available in "real-time" for broker modules and users.

Job Submission performs job submission to GT2 or GT3 systems depending of user criteria and job characteristics. It receives a user criteria and RSL from the user side. To select the appropriate job from local queues the scheduling policy is evaluated.

Job Monitoring controls job status changes and stores their history. It also performs job rescheduling when appropriate (e.g. when a resource has fallen). To do this, some interactions between resource monitoring and job monitoring are needed.

The API is the responsible for providing a unique point of access to broker services. This API can be used by different clients such as user applications, grid portals or command-line.

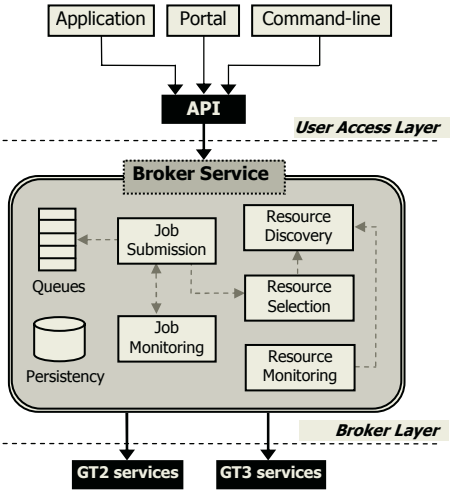


Fig. 1. Overall architecture

The broker design is based on Globus Toolkit as a middleware and as the provider of basic services. Furhtermore, the design is sufficiently extensible to make it easy to adapt the broker to new Globus versions. In order to obtain this, uniform and standard schemes have been used (e.g. GLUE based schema is used for internal resource representation). Recently, some Globus versions have appeared but it is not clear what the evolution of the Grid technology will be like. At present, the Globus project is working on implementations based in Web Services technology, e.g. Web Service Resource Framework (WSRF). These new technologies can be very useful but is very important to keep the compatibility with systems based on previous Globus versions and to give support to its users. There are a lot of projects related to different topics developed on top of

GT2, e.g. DataGrid [18], GridLab [15] or GRID SuperScalar [19]. More detailed description of our broker architecture is presented in the following subsections and more information can be found in [20].

3.2 Job Description and User Criteria

To describe a job a RSL is required and a user criteria is optional. We do not extend RSL schema in order to simplify files and separate concepts. A user criteria is XML-based and specifies basic parameters. A user criteria is composed of several attributes organized in three categories: Memory&Processor, Filesystems&OS, and Others. Each attribute is composed of various elements as shown in Fig. 2:

- *Name*: name of the attribute (e.g. RAMAvailable, ClockSpeed, OSName, etc.)
- *Type*: attribute values can be STRING or INTEGER
- *Operator*: if the attribute type is STRING the possible operator is "==" (identical strings) and if it is an INTEGER attribute possible operator are "==", "<=" or ">="
- *Value*: value of the attribute (corresponding to its type)
- *Importance*: There are two types of attributes, HARD and SOFT attributes. A HARD attribute is a requirement for resources and must be accomplished. However, a SOFT attribute is a recommendation for choosing between all resources that accomplish their requirements.
- *Priority*: this element is considered only in SOFT type attributes in order to obtain a ranking of resources according to the user criteria. The obtained rank value can be useful for later policies evaluation.

```
<?xml version="1.0" encoding="UTF-8"?>
<CRITERIA>
 <Memory-Processor>
    <Attribute Name="RAMAvailable" Operator="&gt;=" Value="100" Type="INTEGER" Importance="HARD" Priority="1" />
    <Attribute Name="VirtualAvailable" Operator="&gt;=" Value="250" Type="INTEGER" Importance="SOFT" Priority="3" />
    <Attribute Name="ClockSpeed" Operator="&gt;=" Value="500" Type="INTEGER" Importance="SOFT" Priority="7" />
    <Attribute Name="LoadLast15Min" Operator="&lt;=" Value="45" Type="INTEGER" Importance="SOFT" Priority="10" />
 </Memory-Processor>
 <FileSystem-OperatingSystem>
    <Attribute Name="AvailableSpace" Operator="&gt;=" Value="600" Type="INTEGER" Importance="SOFT" Priority="7" />
    <Attribute Name="OS Name" Operator="==" Value="Linux" Type="STRING" Importance="HARD" Priority="1" />
 </FileSystem-OperatingSystem>
 <Others>
    <Attribute Name="Total CPUs" Operator="&gt;=" Value="4" Type="INTEGER" Importance="SOFT" Priority="1" />
    <Attribute Name="MaxQueueTime" Operator="==" Value="3600" Type="STRING" Importance="SOFT" Priority="1" />
 </Others>
</CRITERIA>
```

Fig. 2. User criteria example

3.3 Policy Management

As well as basic brokering functions (resource discovery, job submission, etc.) dynamic management of policies and the implementation of the necessary

mechanisms to support them are important subjects in the design of this broker. The selection of the better job and better resource for a given configuration is an optimization problem with NP-Complete solution. In order to reduce and divide the complexity, the broker works with two kinds of basic policies, one for a job scheduling and another for resource selection. Furthermore, beyond job scheduling and resource selection policies, a meta-policy is offered, which can be implemented with genetic algorithms or other optimization methods. The evaluation process of policies is shown in Fig. 3 and consists of three phases. First an initial evaluation of the job scheduling policy is performed and then, for each job selected, the resource selection policy is evaluated and finally the meta-policy evaluation is performed. A meta-policy evaluation consists of choosing the best job to be executed and the best resource for the execution from the data structure obtained from the evaluation of the previous policies. This data structure is a matrix corresponding to the set of jobs obtained in the first step and for each of them a set of resources obtained in the second one.

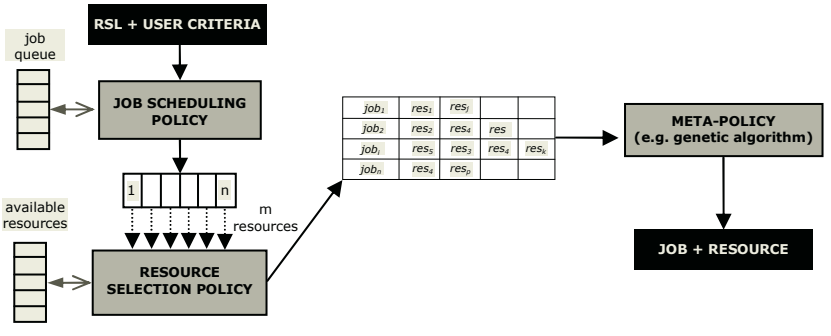


Fig. 3. Policies evaluation schema

Some policies implementations for the broker are outlined next. For job scheduling FIFOjobPolicy (First In First Out), REALTIMEjobPolicy (minimizes REALTIME=deadline time-estimated time of job finalization), EDFjobPolicy (Earlest Deadline First). For resource selection RANKresPolicy (resource selection based in the greatest rank obtained from the resource filtering process), ESTresPolicy (Earlest Starting Time, based in the estimated waiting time for a job in a local queue). User criteria can be used in some evaluation policies like RANKresPolicy.

In order to obtain dynamic policy management we propose a design based in generic interfaces for each kind of policy. Then, the mechanism is ready for a policy evaluation independently of its implementation.

Dynamic management of policies allows them to be managed by the user. Considering that several instances of the broker can exist, it is possible to have broker instances with different policies at the same time. To manage policies from the user's side some interfaces are available to examine the established policy, to change it and to see what policies are available.

3.4 Job Scheduling and Management

When a job is submitted, it is automatically queued in the local system. Periodically the resource broker tries to schedule all jobs according to the established policies. When a machine fails, all the jobs running in that machine are rescheduled through another local queue called retrying. This queue is of higher priority than the submit one in order to prevent inanition situations. Any submitted job is scheduled until all jobs from the retrying queue are managed.

The main issues of job management are job submission, cancellation, monitoring and termination. In order to submit a job, a RSL is required and optionally a user criteria file. The RSL can be a traditional RSL or XML based RSL-2 because the resource broker is compatible with both GT2 and GT3. If the RSL used is the traditional one this job could be executed in a GT2 or GT3 resource indifferently. If RSL-2 is used, the job can only be executed in a GT3 resource because no RSL-2 to RSL parser exits yet. Details about user criteria are shown in subsection 3.2.

To submit a job to a certain resource the Globus GRAM API is used. Callbacks are managed with the GRAM interface responsible for status changes. To decide which is the appropriate resource for a job execution the resource selection policy is evaluated over the resources obtained previously from the resource discovery module.

In the job monitoring process the resource broker is looking for notifications and callbacks to control the job status. In addition, the job history is kept in order to know what is happening and what happened during the job life. In job history some information is considered, such as date, time, operation and other details. In order to preserve data persistence of submitted jobs a recovery file is saved with the necessary job information to resubmit them. In case the resource broker machine crashes, when the resource broker is restarted, all jobs in the recovery file are rescheduled for execution.

In the implementation of this resource broker we encountered some problems related with the Globus APIs, in particular with GRAM. Globus infrastructure adds overhead in job submission and GRAM interfaces are not compatible with different Globus versions at the same time. Thus, it was necessary to implement different interfaces and objects in the job management to give support to GT2 and GT3. Furthermore, Globus client APIs are designed to be used only from the final user side, and we encountered some problems in job submission from a Grid Service. It was therefore necessary to make some changes in code and correct some bugs.

3.5 Resource Management

Our resource broker has a generic and unique representation of resources based in GLUE schema. Therefore, we can use only one internal representation for GT2 and GT3 resources and we can make some decisions independently of the Globus version of resources. The main attributes for this resource representation are general information such as Globus version, hostname or #CPUs, main memory info, operating system, processors info, processors load, file systems info and running jobs.

In order to simplify the resource discovery process we used a uniform representation for resource servers called *Global Grid Resource Information Server*

(GGRIS). In this representation we can specify a MDS GIIS, a GRIS or a GT3 Index Service. From these resource servers the resource broker can obtain resources and resource details in the representtation previous shown.

Resource information is dynamic and the only required functionality to maintain persistent is the GGRIS information. Due to this, we use an XML file with a list of available resource servers. Depending on the server type, different information is needed for specifying its location. For GT3 servers only the Index Service GSH location is needed. However, for GT2 GRIS or GIIS servers, the hostname, port and baseDN are needed.

Resource monitoring updates local data about resources by calling the resource discovery module continuously. In order to detect when a resource has failed the resource broker compares current available resources with the previous data before updating the list of resources. In the case of a detection of one or multiple resource falling, this module interacts with job management modules rescheduling their jobs.

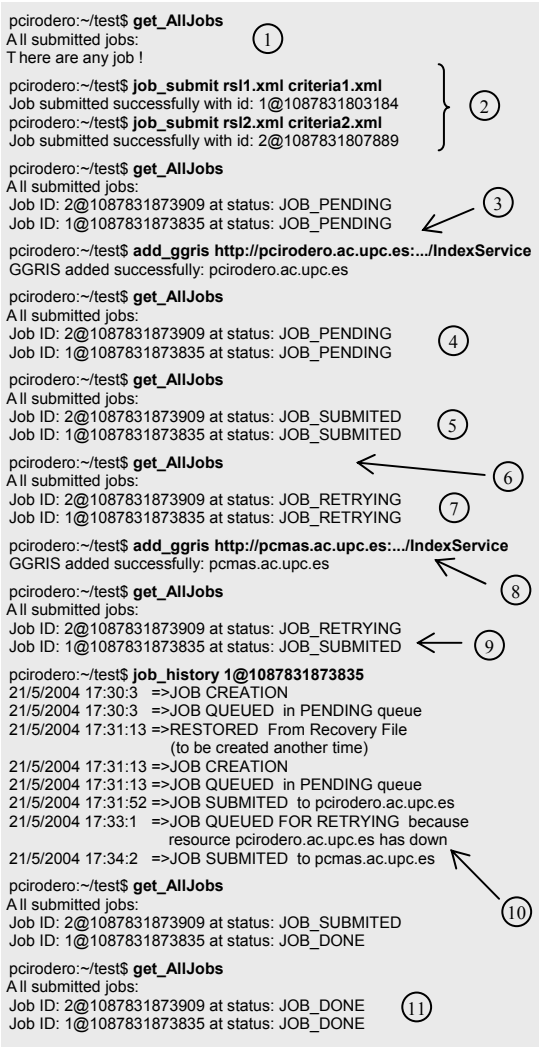


Fig. 4. Execution of some broker commands

In both GT2 MDS and GT3 Index Service we use the scripts provided by Globus. The GT3 Index Service is a useful mechanism for indexing data but in some cases the scripts provided by Globus are not powerful enough and the provided data is not updated. In general, there is a lack of information about local resource management and performance monitoring. For instance, the behavior of applications is very useful information to make scheduling decisions with coordination. Consequently, it is difficult to give good support to HPC resources with Globus infrastructure.

4 Experimental Results

We present results to demonstrate the functionality of the broker. Since the main problem of GT3 is the overhead, we also present some performance results.

4.1 Behavior Analysis

In order to illustrate the broker behavior we are going to use an example shown in Fig. 4. In that example we can find several circumstances and actions related with the broker. Next, we explain each event and what decisions the broker system takes.

Initially, no resource is available and no job is submitted. Next in (2), some jobs are submitted to the broker and are queued to the pending queue. In (3) the broker falls. When the broker is restarted we add a computational resource in order to execute submitted jobs. Then, in (4) the broker retrieves previous submitted jobs from the recovery system and queues them again in a local queue. In (5) jobs are submitted to the available resource and begin their execution. Suddenly, the resource which was executing jobs (pcirodero) falls, in (6). So, in (7) all jobs that were submitted in pcirodero are queued to retry their execution. Now there is no resource available, so in (8) we add a new computational resource to allow job execution. Afterwards jobs begin their execution on pcmas, in (9). In (10) we can see a job history and how all events have happened. Finally, in (11) all jobs finish their execution on pcmas.

4.2 Performance Analysis

In order to study the broker and system performance, we instrumented the broker through JIS and JACIT [21]. JIS enables us to instrument Java classes and to obtain some traces which can be visualized and analized with Paraver [22]. First, we present results obtained from an execution of a minimum job in a GT3 resource. With these results we can approximate various types of overhead such as Globus, the broker or communications overhead. In Fig. 5 a trace obtained from the execution of two minimal jobs is shown. In Paraver traces we can see the time on the X axis, each row

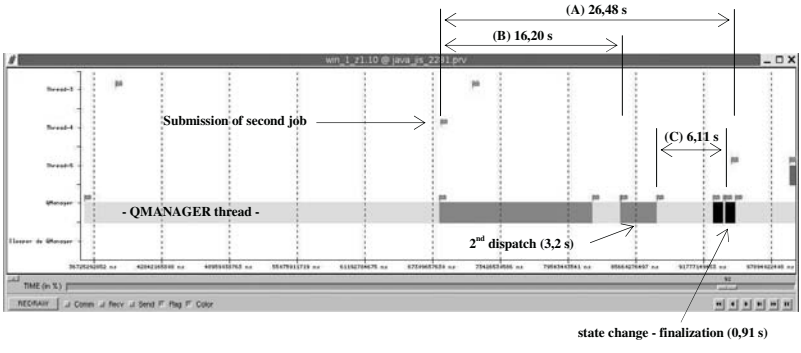


Fig. 5. Trace obtained in the minimal execution

represents a thread, colors represent different states and flags are events. We can see some events (such as submission requests, job dispatching actions, finalization notifications and state requests) or the spent time for each one.

(A) is the elapsed time between the reception of the job submission and the moment of its conclusion (including the state change, total process).
(B) is the queuing time of the job until its submission to a specific resource.
(C) is the elapsed time between the job submission to a specific resource and the notification of its conclusion.

Then, considering the data obtained in this test, we can obtain some numbers relating to the overhead. Broker overhead=61%, Globus overhead=16%, the other overhead is not relevant. **Total overhead=77%** but this is only the result obtained from a concrete execution of a minimal job.

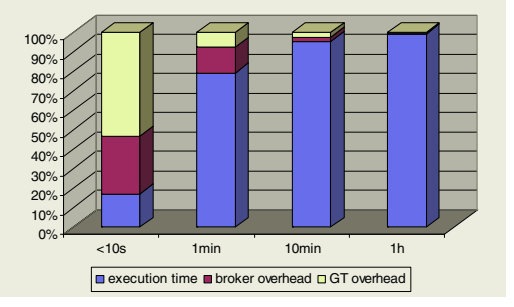


Fig. 6. Distribution of time in tests

Now we are going to present average results obtained from the execution of several tests of different duration. In Fig. 6 results of those arithmetic jobs are shown, in short executions we obtain big overhead but from a job of a minute duration, we obtain acceptable values. Then we can say the broker is the broker is suitable enough for Grid oriented applications[1] in terms of overhead.

5 Conclusion and Future Work

In this paper, we have designed and implemented an OGSI-compliant Grid resource broker. Our resource broker performs resource discovery and management, scheduling and hides the underlying complexity of Grid resources from Grid users. It is compatible with both GT2 and GT3 services and is designed as an extensible and modular way to be easily extended and become compatible with future Globus versions. Moreover, the proposed resource broker considers dynamic policies management. To achieve these goals, the resource broker implements powerful mechanisms to allow users to manage policies using a Grid Service based interface, API, and client. Through experimental evaluations, we have successfully shown that the resource broker system behavior and its performance are satisfactory for Grid oriented applications.

For future work we are seeking to improve our resource broker with greater robustness, check pointing, job migration and so on. We plan to add low level interaction with local queuing systems in order to choose the best approaches in Grid environments especially for parallel applications. Moreover, we need to implement

[1] In this paper we do not consider results from the overhead of data transport and management.

complex meta-policies and new policies based on prediction concepts. We wish to implement support for GT4 when this is stable. Finally, we plan to construct more scalable systems.

With the experience gained from developing a broker on top of Globus Toolkit, we have found some deficiencies. First, we believe that APIs need to be improved to give better support for developers; currently Globus APIs are designed to be used for final users. Resources should be managed in a more effective way. A useful middleware should provide good monitoring tools and enable communications between the middleware and the local environment. Finally, the Globus Toolkit should be improved to reduce its overhead, and we hope future versions will be better.

Acknowledgments

This research has been supported by the Spanish Ministry of Science and Technology under contract TIC2001-0995-C02-01, and the European Union project HPC-Europa under contract 506079.

References

1. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of High Perfomance Computing Applications, 15(3):200-222. 2001.
2. "The Globus Project (Globus Alliance)", http://www.globus.org
3. M. Litzkow, M.Livny, and M. Mutka, "Condor – A Hunter of Idle Workstations", Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS 1988), January 1988, San Jose, CA, IEEE CS Press, USA, 1998
4. "Unicore Project", http://www.unicore.org
5. I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Open Grid Service Infrastructure WG, Global Grid Forum, 2002.
6. Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz, "Grid Resource Management, State of the Art and Future Trends", Kluwer Academic Publishers, 2004
7. "Globus Security Infrastructure", http://www.globus.org/security
8. "Resource Management: GT2 GRAM and GT3 GRAM",
9. http://www-unix.globus.org/developer/resource-management.html
10. "Globus Data Management Services",
11. http://www-unix.globus.org/toolkit/docs/3.2/datamanagement.html
12. "Information Services in the Globus Toolkit", http://www.globus.org/mds
13. F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks", Proceedings of Supercomputing'96, 1996
14. D. Abramson, R. Buyya, and J. Giddy, "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker", Future Generation Computer Systems. 18(8), 2002
15. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids". Proceedings of the Tenth International Symposium on High Performance Distributed Computing, IEEE CS Press, August 2001

16. B. Chapman et al, “EZ-Grid Resource Brokerage System”, http://www.cs.uh.edu/~ezgrid/
17. “GridLab, A Grid Application Toolkit and Testbed”, http://www.gridlab.org
18. “The GridWay Project”, http://asds.dacya.ucm.es/GridWay/
19. Young-Seok Kim, Jung-Lok Yu, Jae-Gyoon Hahm, Jin-Soo Kim, and et al. “Design and Implementation of an OGSI-Compliant Grid Broker Service”, Proc. of CCGrid 2004
20. “The DataGrid Project”, http://www.eu-datagrid.org
21. Rosa M. Badia, Jesús Labarta, Raül Sirvent, Josep M. Pérez, José M. Cela, and Rogeli Grima, “Programming Grid Applications with GRID superscalar”, Journal of Grid Computing, January 2004
22. Ivan Rodero, Julita Corbalán, Rosa M. Badia, Jesús Labarta, “Providing a Resource Broker for eNANOS Project”, Technical Report UPC-DAC-2004-43, Tech. U. of Catalonia, 2004
23. Jordi Guitart, Jordi Torres, Eduard Ayguadé, José Oliver, and Jesús Labarta, “Java Instrumentation Suite: Accurate Analysis of Java Threaded Applications”, 2nd Annual Workshop on Java on High Performance Computing, Santa Fe, New Mexico, USA, 2000.
24. Paraver, http://www.cepba.upc.edu/paraver/

GridARM: Askalon's Grid Resource Management System*

Mumtaz Siddiqui and Thomas Fahringer

Institute for Computer Science, University of Innsbruck,
Technikerstrasse 13, A-6020 Innsbruck, Austria
{Mumtaz.Siddiqui, Thomas.Fahringer}@uibk.ac.at

Abstract. The emergence of Grid computing has accentuated the need of an adaptable, scalable and extensible resource management system. In this paper we introduce GridARM system which renders the boundaries of resource brokerage, virtual organization wide authorization and advanced reservation, and represents a scalable and adaptive Grid resource management as a middleware infrastructure. The GridARM system provides mechanisms for Grid resource discovery, selection and allocation along with resource requestor and provider interaction. Experiments are presented that demonstrate the effectiveness of our approach.

1 Introduction

With the emergence of distributed computing, and the 'always on' environment of the computing elements; the computing services become scalable, extensible and environment independent. This results in a high performance computational environment composed of diverse resources spanning the entire Internet and multiple administrative domains. The new discipline called *Grid Computing* intends to make high performance computational resources available to anyone. An effective Grid Resource Management System (GRMS) is required for the provisioning and sharing of resources while keeping autonomy of their environment and geographical location.

In contrast to traditional resource management system, GRMS has to balance global resource sharing with local autonomy, by dealing with heterogeneous, shared and variant resources distributed under different trust domains, addressing issues of multiple layers of schedulers and working with system participants having inconsistent performance goals and assorted local and global policies.

In the Grid computing literature, Grid job scheduling is represented and treated as part of the Grid resource management, therefore in most of the existing Grid enabled systems, meta scheduling is integrated with resource management and the terms *Grid job scheduling* and *Grid resource management* are used interchangeably. Now the Grid computing has been evolved enough to redefine and redesign its components so that they can be used as self comprised building

* This research is supported by the Higher Education Commission of Pakistan.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 122–131, 2005.

blocks in GRMS. Resource broker is one of the important GRMS components, which is manual or semi manual in existing systems. An automatic resource broker is essential for a stable and successful Grid computing infrastructure.

A GRMS must provide *Resource discovery and selection mechanism* which performs persistent resource state and capacity checking, by discovering and matching resources, *Capability check mechanism* which performs resource selection based on dynamic information, *Resource allocation* with advance reservation and co-allocation and finally *Resource requester and provider interaction* for negotiation and notifications.

To our knowledge no widely deployed single GRMS supports these functions. Attribute based resource description and resource matching available in existing systems is unsuitable for ever evolving Grids. Virtual Organization (VO) wide authorization, advance reservation and Co-allocation are still illusions for the Grid users. We propose a new approach to flexible resource management for Grid computing system which provides the management functions mentioned above. The goal of our work is to provide an effective, efficient, extensible and adaptive GRMS based on off-the-shelf technologies while making it capable to adapt new emerging technologies.

This paper presents a design architecture and work-in-progress prototype implementation of *GridARM* (Askalon's [14] Grid Resource Management) System, a new architecture for Grid resource management, resource control and resource provisioning across sites and administrative domains. It provides automatic resource brokerage, VO-wide fine-grained authorization, advanced reservation and negotiation between a potential client and resource provider. The automatic broker was necessary not only because of its usability, efficiency and low cost but also because users don't have time to make selection between alternative choices. VO-wide authorization and user profiling mechanism reduces involvement of local site administrators and even the user itself.

The rest of the paper is as follows. In Section 2, we describe general Grid resource management architecture and define basic mechanisms to provide an automatic resource brokerage system. Section 3 is about the client-GridARM interaction mechanism. In Section 4, we described our experiences about the proposed system, and examined its performance in a networked environment. Related work is presented in Section 5. Finally we summarize our conclusion about the proposed system and discuss future work in Section 6.

2 GridARM Architecture

The GridARM system is dynamically extensible, scalable and adaptive in which new protocols and tools can easily be integrated without suffering from system downtime and expensive code reorganization. In contrast to existing work which is based on manual brokerage we propose an automated brokerage in this system. This automation is required especially for Grid enabled workflows and execution environments where the brokerage process acts as a middle tier between Meta-scheduler and other Grid enabled components like Grid enabled resources and

services. The brokerage process is responsible to discover and allocate suitable resources for the Meta schedulers.

We are developing *GridARM* system which is WSRF [11] complaint. The system consists of four persistent and distributed Grid-enabled services called Discovery, Authorization, Reservation and Broker.

As shown in the Fig. 1, the Broker service is a gateway to the GridARM sytem. It is a configurable and customizeable WS-Resource which works with one or more other GridARM services. An important feature of the Broker is its ability to recursively discover Grid resources by interacting with the other distributed GridARM brokers.

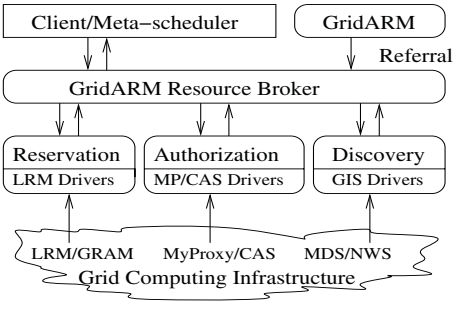


Fig. 1. GridARM system architecture

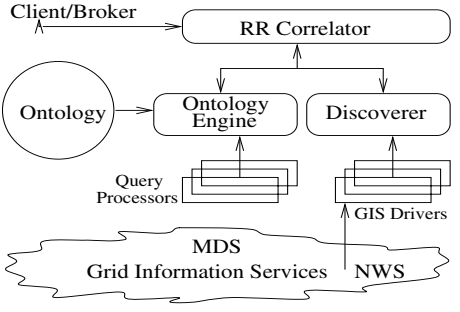
The *Discovery service* provides resource discovering and matching capability, mainly to the Broker. It can be configured with one or more Grid Information Services (GIS).

Authorization and *Reservation* services provide resource authorization and advance reservation capabilities respectively. Authorization is based on the Grid Security Infrastructure (GSI) [3] and works in coordination with My-Proxy [7] and Community Authorization Service (CAS) [2]. The Reservation service provides capabilities like advanced reservation and negotiation for reservation with the resource provider, based on time and cost model and other constraints set by the both parties.

Each *GridARM service* is configured to have one or more drivers to the resource specific modules like GIS service. In the following sections we describe GridARM services in detail.

2.1 Discovery Service

A resource discovery mechanism mainly provided to the *Broker* service is based on a set of information and monitoring services. It discovers and provides an optimal resource *ensemble* along with solicited specification congregated from the underlying information services. We are currently working on a *Grid resource description language*, a language to be used for describing Grid resources in an ontological way. The internal structure of a discovery service is illustrated in Fig. 2. It consists of a *Request-Resource Correlator (RRC)*, *Ontological Engine (OE)* and *Resource Discoverer (RD)*. *Correlator* receives a request, checks its integrity, correlates it with the resources and returns the result back. It interacts with *OE* and the *RD* for request transformation and resource discovery. A client can subscribe in the discovery service by registering resource requirements or preferences in the *RRC* and receives notification from it when a matching resource joins the Grid and observed by the discovery service.




```
<ResourceRequest>
 <resources>
  <totalNodes constraint="exact"> 15 </totalNodes>
  <constraints>
   <cpuCount constraint="min">4</cpuCount>
   <cpuAvailable constraint="min">2.5</cpuAvailable>
   <osType flavor="any">Linux</osType>
   <platform>sparc</platform>
  </constraints>
 </resources>
</ResourceRequest>
```

Fig. 2. **A**: Internal structure of a discovery service **B**: A simple resource request with both static and dynamic attributes

Ontological Engine selects appropriate *Query Processor (QP)* and transforms the request into resource *filters* for the registered information services, based on which resources are discovered and selected. The transformation is done in two steps: First, the compound request is split into multiple but simple requests, for example by separating static and dynamic attributes of a request and making two different GIS-specific requests. Second, it transforms requests from generic format to GIS specific *filters*. For instance, a generic request given in Fig. 2 is transformed into LDAP filter for the MDS2 [4] driver.

The *RD* component discovers resources and their specification based on the *filters* provided by *OE* and congregates all found resources along with their specification into a unique GIS-independent generic format. Finally it returns the result back to the *Correlator*.

Discoverer drivers and *query processors* are loaded dynamically based on the GIS and query *type* respectively. The default GIS type is *mds2* and query type is *generic*. A client can make a query for resources based on both static and dynamic attributes. For instance, a resource request given in Fig. 2, contains both static and dynamic attributes and discovery service uses MDS to collect static information like *OsName* and *Platform* and NWS [5] to collect dynamic information of resources such as *AvailableCpu*.

2.2 Reservation Service

Reservation is an undertaking by the system that an application will receive a certain level of service from its resources. The reservation service provides this functionality by supplying *instant* as well as *advanced reservation* of underlying Grid-enabled resources. The architecture is flexible enough to work with different local reservation managers for example Maui [6]. But our aim is to provide a new VO wide reservation manager which is consistent with other Globus Toolkit (GT) based components like CAS [2]. This service is used to interact with a single resource, whereas Co-allocation of multiple resources is handled by the Broker service which also acts as a *Co-allocation manager*.

The reservation service provides high level methods to *create*, *modify*, *bind*, *cancel*, monitor and *verify* a resource *reservation* instance. Verification is re-

quired to be performed before acquiring a reserved resource by a job submission component. Also, the service can be used to look ahead for the advanced reservation of a resource by employing its *lookahead* method. The Local Reservation Managers (LRM) can be registered and managed dynamically.

The essential attributes of a reservation instance are *LRM contact*, *reservation mode*, *start time*, *end time*, *duration* and *constraint*s. The reservation mode can be a *single phase* or a *two-phase committable*. In two-phase mode, once a reservation is created, it remains on hold for a configurable *duration*. If reservation is not committed within that time interval, then it is cancelled automatically. The reservation, which is not committed with in specified *time-frame*, represents a soft allocation of resource as described in Section 2.4. The optional *end time* attribute can also be provided for flexibility. The reservation is accepted if it can be started any time after *start time* and finished any time before *end time*.

A special *Constraint-based Advanced Reservation (CAR)* can be created based on resource constraints instead of hard coded resource manager contact. In CAR, the LRM contact *linking* is deferred until *bind* time. Reservation service ensures that minimum required resources which fulfill the CAR's constraints should be available during that *time frame*. In this way the required QoS is ensured.

A *Ticket*, which is granted by the service after making a reservation, is a reference to a reservation instance. It embraces a unique *reservation id*, user's *security principal* and resource *access point*. Once a reservation has been made, all future interactions, like monitoring for its status, modification, cancellation and resource acquisition can only be done by producing a valid *ticket*. Before performing any task, a user *credential* is produced or retrieved from the Authorization service. The reservation is not possible if a user possesses invalid credential or a given policy does not permit a reservation. A policy is described in the form of constraints and used with authorization information to provide enhanced quality of service. One can specify a list of users or groups allowed to use a reservation ticket in subsequent operations. The integrity is ensured by signing the policy with trusted entity.

2.3 Authorization Service

The Globus system lacks a middleware-based authorization. It uses local resource mechanism for authorization by mapping a Grid user to a local identity which also serves as an access control check. This scenario has several shortcomings such as scalability, lack of expressiveness, and consistency between the policies of different sites. In order to overcome these shortcomings GridARM proposes an authorization mechanism, in which, the Grid users and sites are registered in a middleware service, which maintains user profiles, proxy credentials, policies and preferences. The Grid sites are registered by creating a local identity representing the VO or community.

This service grants authorization to a user by verifying its access rights for a particular *resource ensemble* and provides a *restricted proxy credential* to the user. A restricted proxy credential can only be used during the reservation time-

frame. Our authorization service uses Globus My-Proxy [7] as a credential repository, and we plan to integrate Community Authorization Service (CAS) [2] in our approach.

2.4 Broker Service

The *Broker* service works as a gateway to the *GridARM* system. It makes an efficient and smart use of the other services, which can be registered and managed dynamically. It also works as a *Co-allocation manager* and performs advance reservation of multiple resources on request. Its role in the overall system is illustrated in Fig. 1. High level interfaces are provided for resource selection, allocation and management. These interfaces include methods like *select*, *allocate*, *confirm* and *release* etc. Selection operation results in a *resource ensemble* based on the resource request, whereas allocation operation results in a *reservation ensemble* or reservation *ticket*. The input to the broker, provided mainly by Grid scheduler, is examined and an appropriate and optimal operation is performed.

The allocation of resources can be a result of an *interactive transaction* by following *select-allocate-confirm* cycle in steps, or it can be an *atomic transaction* by making a direct confirmed reservation. An allocation of resources without confirmation is a *soft allocation*. A soft allocation is one in which allocated resources will be available to new clients only if (1) they are explicitly released or (2) they are not used within certain configurable *duration* of time. Once reservation is confirmed it becomes a *hard allocation* that means the allocated *resource ensemble* remain dedicated to a client during the reservation *timeframe*.

If a broker could not find suitable resources, it can refer its clients to another broker service, or it can be configured to work in a recursive mode to retrieve required resources by interacting with the *remote* brokers.

Once a reservation ensemble is created, the Broker instantiates a resource *Ensemble Manager (EM)* on one of least loaded GridARM-enabled hosts. *EM* is responsible for further coordination and negotiation with client on behalf of resource providers. It provides comprehensive functionality for editing and managing a reservation ensemble. Also *EM* monitors its members while they are being used by the client and ensures that resources are working according to the constraints specified at the time of reservation. A node failure, if occurs, is notified to the job submission system.

The GridARM system works based on GSI [3] provided by the Globus. A client can interact with system via frontend broker service by providing its own proxy credential supplied directly or through My-Proxy [7]. If an inter-VO-referral based recursion is performed, then it would be possible that a requesting user is not part of a referred VO. In this case user credentials are replaced with referral or 'remote' broker's credential and a chained authorization is performed to make it possible. The management interfaces of GridARM services are implemented with message level security, so that an unauthorized Grid user could not make changes in the configuration.

3 Interaction Mechanism

Advanced reservation and Co-allocation of a *resource ensemble* is a multi transactional process, which is simplified by providing a *Proxy Resource Ensemble(PRE)* in the response of a resource request. The broker *execute*s a request by selecting a *resource ensemble*, instantiating an *Ensemble Manager (EM)*, and returning back a *proxy (PRE)*. The *PRE* being a mobile agent, is downloaded to a client machine and used as a stub of the resource ensemble for further interaction with the *GridARM* system. It hides authorization mechanism and optionally physical resources from the client. This is done by providing a *login mechanism*, in which the user along with reservation *ticket* is verified, and the *user credential* is replaced with *restricted community credential*. The client presents community credential to the resource ensemble before submitting a job. We plan to integrate a policy evaluation mechanism in the PRE in order to provide a fine-grained user authorization by the PRE.

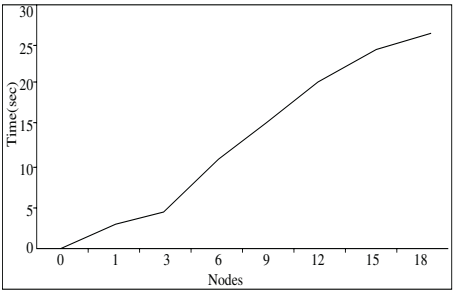
4 Experiments

The Globus toolkit provides a decentralized scheduling model in which new components can be integrated. An unofficial release of GT4 which consists of core implementation of WSRF [11] was introduced in the beginning of 2004. WSRF is a new model on which Grids are to be built. The GridARM system is WSRF complaint and uses mechanisms like subscription/notification and service lifetime management. We have deployed the system in ZID-Grid, University of Innsbruck, which consists 13 Grid sites, one with 15 Solaris machines and 12 with 141 Linux PC boxes. GT2 is installed on all Grid sites whereas GT4 core is installed only on Solaris machines. Apart from site specific services like GRAM [1], we have installed NWS [5] and MDS2 [4] with the Glue schema. Both MDS and NWS cover all ZID-Grid sites. All the machines involved in the experiment were located on a lightly loaded network with a maximum latency between two computers of about 2 milliseconds.

A resource request can be a simple or compound request. A *simple request* could be an attribute-based request for the selection of resources or a reservation request for already selected resource. A *compound request* is one in which multiple operations are requested atomically. In the following sections we describe different experiments conducted in the deployed Grid infrastructure.

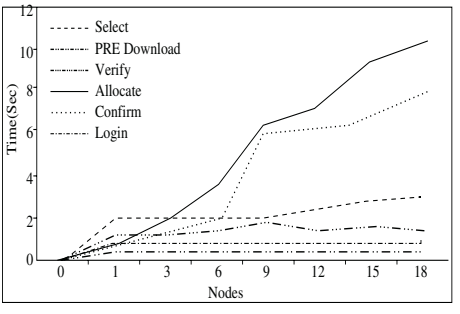
4.1 Atomic Transaction

A compound request shown in Fig. 3 is made to the system for resource allocation. The request consists of both resource and reservation description. In this scenario, the broker service performs resource *selection*, *authorization*, *allocation* and *confirmation* cycle as a *single* transaction. To determine the cost of requested operation, we timed a series of requests varying both the total number of required resources and their attributes. We measured the time for the request




```
<ResourceRequest>
 <type> compound </type>
 <resources negotiation="no">
  <constraints>
   <cpuCount constraint="min">3</cpuCount>
   <cpuCurrent constraint="min">1.5</cpuCurrent>
   <osType flavor="any">Linux</osType>
   <platform>intel</platform>
  </constraints>
 </resources>
 <reservation negotiation="yes">
  <startTime format="YYMMddhhmm" constraint="exact">
   0410151230
  </startTime>
  <duration unit="minutes">150</duration>
 </reservation>
</ResourceRequest>
```

Fig. 3. A: GridARM system latency for resource reservation, time taken for atomic transactions for the given number of nodes/CPUs **B**: Atomic transaction: A compound request for a resource ensemble selection and confirmed allocation



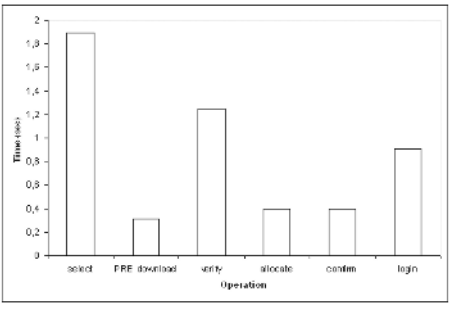


Fig. 4. A: Time variation of GridARM system functions **B**: Breakdown of times spent in processing a resource allocation request for a single resource

by starting a timer in the client program immediately before invoking the broker and then stop this timer on successful login to the resource ensemble through proxy *PRE* (See Section 3). The result of this experiment is shown in Fig. 3. The graph shows how the time for request brokerage varies as the number of resources changed. A further breakdown of the time spent in different GridARM operations is given in Fig. 4, which shows the cost of each operation.

4.2 Interactive Transaction

In this experiment, the *compound request* shown in Fig. 3, is broken down into two requests, and the brokerage is performed in steps by implying *execute*, *verify*, *allocate*, *confirm* and *login* functions in a sequence by the client. The interactive transaction is useful in case when a client wants to perform some intermediate tasks. A simple use case could be as follows: A Metascheduler first reserves the resource ensemble and then confirms after interacting with a performance predictor, that may evaluate a Grid resource in the meantime. A client can also call *lookahead* function for the reservation of the resource ensemble before making a confirmed reservation.

A breakdown of the time spent in different system functions is given in Fig. 4. The authorization (*verify* + *login*) and discovery (*select*) services are very con-

sistent and economical. Most of the time is consumed by *allocate* and *confirm* functions of the reservation service. This is due to the fact that currently a separate request is made for each resource in the resource ensemble. We plan to enhance the service by adding the functionality in which an entire resource ensemble could be handled with a single request. As shown in the Fig. 4(right), *allocate* and *confirm* functions collectively take less than one second, therefore there will be a significant improvement in the system performance after having the enhanced functionality.

5 Related Work

In the domain of GRMS, numerous projects and tools are available, but most of them do not provide the required level of resource management. This pervasive domain needs to split down further in more self contained and adaptable sub domains. Most of the existing Grid enabled systems try to address resource brokerage, job scheduling and monitoring under the same integrated scenario. It works, but it's not scaleable and adaptable. The resource broker in the Globus system is missing. A few Grid systems like Condor [8], Legion [9], GridLab [16], European Data Grid [10], Nimrod-G [15] and Maui [6] address GRM but the broker is not a well divulged and concrete module. Also none of these systems addresses resource management as a mechanism of the Grid middleware, in which distributed resource brokerage, community-based authorization and advanced reservation are consistent with each other.

A distributed resource management architecture that supports advance reservation and Co-allocation is described in [12] but the modification proposed in the local resource management is an overhead. Ontology based resource matching proposed in [13] simplifies resource matching, but community based authorization and reservation has not been addressed. The Global Grid Forum (GGF) is actively working on devising new standards in different areas of resource management. The GridARM system will adopt GGF standards once fully specified.

6 Conclusion and Future Work

Unleashing the power of Grid infrastructures is a complex and tedious task without a sophisticated resource management system. The focus of this paper is to render the boundaries of resource brokerage, community wide authorization and advanced reservation mechanism. The paper proposes a modular and dynamically extensible Grid resource management architecture, which fills the gap between Meta-scheduler and the Grid infrastructure. The GridARM system makes the use of the Grid resources simple and efficient with the help of VO-wide authorization and reservation mechanism. Our aim is to make the system fully consistent with the Grid forum recommendations. We are evaluating different related technologies which can be exploited to make the system more functional and consistent with emerging web technologies.

The GridARM system automates the process of the Grid resource management, but its own management and VO-wide deployment is manual. The plan for the future enhancement of the system is to make it fully automated.

References

1. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. *A Resource Management Architecture for Metacomputing Systems.* Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, po. 62-82, 1998.
2. L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke. *A Community Authorization Service for Group Collaboration.* IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2001.
3. Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. *A Security Architecture for Computational Grids.* In Fifth ACM Conference on Computers and Communications Security, November 1998.
4. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. *Grid Information Services for Distributed Resource Sharing.* Tenth IEEE International Symposium on High-Performance Distributed Computing(HPDC-10), IEEE Press, Aug 2001.
5. Rich Wolski, Neil Spring, and Jim Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing* Future Generation Computing Systems Journal, Vol 15, 5-6, pp. 757-768, Oct. 1999.
6. *The Maui Scheduler* home page. http://maui-scheduler.mhpcc.edu
7. J. Novotny, S. Tuecke, V. Welch. *An Online Credential Repository for the Grid: MyProxy.* Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
8. *Condor project homepage.* http://www.cs.wisc.edu/condor/
9. Steve Chapin, Dimitrios Katramatos, John Karpovich, Andrew Grimshaw. *The Legion Resource Management System.* Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99)
10. B. Sagal. *Grid Computing: The European DataGrid Project.* In IEEE Nuclear Science Symposium and Medical Imaging Conference Lyon, France, October 2000
11. *Web Services Resource Framework.* http://www.globus.org/wsrf/
12. K. Czajkowski, I. Foster, and C. Kesselman. *Resource Co-Allocation in Computational Grids.* In Proc. of the 8-th IEEE Int'l Symp. on High Performance Distributed Computing, pages 219-228, Redondo Beach, CA, USA, July 1999
13. H. Tangmunarunkit, S. Decker, and C. Kesselman. *Ontology-based Resource Matching in the Grid–The Grid meets the Semantic Web.* Second International Semantic Web Conference, Sanibel-Captiva Islands, Florida, USA, Oct. 2003
14. Thomas Fahringer *ASKALON: A Programming Environment and Tool Set for Cluster and Grid Computing* Institute for Computer Science, University of Innsbruck. http://dps.uibk.ac.at/askalon/
15. R. Buyya, D. Abramson, and J. Giddy. *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, HPC ASIA'2000, China, IEEE CS Press, USA, 2000
16. Allen, G., Davis, K., et al. (2003). *Enabling Applications on the Grid: A GridLab Overview* In International Journal of High Performance Computing Applications: Special Issue on Grid Computing, August 2003.

A Super-Peer Model for Building Resource Discovery Services in Grids: Design and Simulation Analysis*

Carlo Mastroianni[1], Domenico Talia[2], and Oreste Verta[2]

[1] ICAR-CNR 87036 Rende (CS), Italy
mastroianni@icar.cnr.it
[2] DEIS University of Calabria, 87036 Rende (CS), Italy
{talia,verta}@deis.unical.it

Abstract. As deployed Grids increase from tens to thousands of nodes, Peer-to-Peer (P2P) techniques and protocols can be used to implement scalable services and applications. The super-peer model is a novel approach that helps the convergence of P2P models and Grid environments and can be used to deploy a P2P information service in Grids. A super-peer serves a single Virtual Organization (VO) in a Grid, and manages metadata associated to the resources provided by the nodes of that VO. Super-peers connect to each other to form a peer network at a higher level. This paper examines how the super-peer model can be used to handle membership management and resource discovery services in a multi-organizational Grid. A simulation analysis evaluates the performance of a resource discovery protocol; simulation results can be used to tune protocol parameters in order to increase search efficiency.

1 Introduction

Grid computing and peer-to-peer (P2P) computing models share several features and have more in common than we generally recognize. As Grids used for complex applications increase from tens to thousands of nodes, their functionalities should be decentralized to avoid bottlenecks. The P2P model could favor Grid scalability: designers can use P2P style and techniques to implement decentralized Grid systems. The adoption of the service oriented model in novel Grid systems (for example the Open Grid Services Architecture (OGSA [1]), or the Web Services Resource Framework (WSRF) [12]) will support the convergence between the two models, since Web Services can be used to implement P2P interactions between hosts belonging to different domains.

P2P techniques can be particularly useful to manage two key services in Grid information systems: *membership management* (or simply *membership*) and *resource discovery*. The objective of a membership management service is twofold: adding a new node to the network, and assigning this node a set of neighbour nodes. The resource discovery service is invoked by a node when it needs to discover and use hardware or software resources having given characteristics.

* This work was partially supported by the Italian MIUR FIRB Grid.it project RBNE01KNFP on High Performance Grid Platforms and Tools and by the project KMS-Plus funded by the Italian Ministry of Productive Activities.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 132 – 143, 2005.

In currently deployed Grid systems, resources are often owned by research centres, public institutions, or large enterprises: in such organizations hosts and resources are usually stable. Hence, membership management and resource discovery services are efficiently handled through centralized or hierarchical approaches, as in the OGSA and WSRF frameworks. As opposed to Grids, in P2P systems nodes and resources provided to the community are very dynamic: peers can be frequently switched off or disconnected. In such an environment a distributed approach is more effective and fault-tolerant than a centralized or hierarchical one.

Super-peer networks have been proposed [13] to achieve a balance between the inherent efficiency of centralized search, and the autonomy, load balancing and fault-tolerant features offered by distributed search. A super-peer node acts as a centralized resource for a number of regular peers, while super-peers connect to each other to form a network that exploits P2P mechanisms at a higher level. The super-peer model allows for a very efficient implementation of the information service and it is naturally appropriate for large-scale Grids. A Grid can be viewed as a network composed of small-scale, proprietary Grids, called Virtual Organizations (VOs). Within each VO, one or more nodes, e.g. those that have the largest capabilities, can act as super-peers, while other nodes can use super-peers to access the Grid.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the super-peer model, and shows how it can be used in service-oriented Grid frameworks. A discovery protocol based on the super-peer model is proposed and discussed. Section 4 analyzes the performance of the proposed discovery protocol by means of an event-driven simulation framework. The influence of network and protocol parameters on performance indices is evaluated, so that the protocol can be tuned to increase search efficiency. Section 5 concludes the paper.

2 Related Work

P2P membership and discovery services can be classified as using unstructured or structured approaches to search resources. Gnutella [4] is an example of unstructured P2P network: hosts and resources are made available on the network without a global overlay planning. Structured P2P networks, such as Chord [10], use highly structured overlays and exploit a Distributed Hash Table (DHT) to route queries over the network. A DHT is a data structure for distributed storing of pairs (key, data) which allows for fast locating of data when a key is given.

Membership and resource discovery services are also key issues in Grid systems. A centralized or hierarchical approach is usually adopted. The information model exploited in the Globus Toolkit 3 (GT3) the version of Globus built upon OGSA, is based on Index Services [3], a specialized type of Grid Services. Index Services are used to aggregate and index *Service Data*, i.e. metadata associated to the resources provided by Grid hosts. There is typically one Index Service per Virtual Organization but, in large organizations, several Index Services can be organized in a hierarchy. A similar approach is used in the WSRF-based Globus Toolkit 4: ServiceGroup services are used to form a wide variety of collections of WS-Resources, a WS-Resource being a Web service that is associated with a stateful resource.

Today, the Grid community agrees that it is not efficient to devise scalable Grid resource discovery based on a centralized or hierarchical approach when a large number of Grid hosts, resources, and users have to be managed, also because of the heterogeneity of such resources.

Recently, super-peer networks have been proposed to achieve a balance between the inherent efficiency of centralized search, and the autonomy, load balancing and fault-tolerant features offered by distributed search. In [13], performance of super-peer networks is evaluated, and rules of thumb are given for an efficient design of such networks: the objective is to enhance the performance of search operations and at the same time to limit bandwidth and processing load. In [7] a general mechanism for the construction and the maintenance of a superpeer network is proposed and evaluated. A gossip paradigm is used to exchange information among peers and dynamically decide how many and which peers can efficiently act as superpeers.

In [8] both resources and the content stored at peers are described by means of RDF metadata. Routing indices located at super-peers use such metadata to perform the routing of queries expressed through the RDF-QEL query language. Puppin et al. [9] proposed a Grid Information Service based on the super-peer model and its integration within OGSA. The Hop Counting Routing Index algorithm is used to exchange queries among the super-peers and in particular to select the neighbour super-peers that offer the highest probability of success.

3 A Super-Peer Model for Grids

The super-peer model can be advantageously exploited in Grid systems for the deployment of information and discovery services. To maximize the efficiency of the super-peer model in Grids, it is useful to compare the characteristics of Grids and P2P networks.

(i) Grids are less dynamic than P2P networks, since Grid nodes and resources often belong to large enterprises or public institutions and security reasons generally require that Grid nodes authenticate each other before accessing respective resources.

(ii) Whereas in a P2P network users usually search for well defined resources (e.g. MP3 or MPEG files), in Grid systems they often need to discover software or hardware resources that match an extensible set of resource descriptions. Accordingly, while structured protocols, e.g. based on distributed indices, are usually very efficient in file sharing P2P networks, unstructured or hybrid protocols seem to be preferable in largely heterogeneous Grids. Another consequence is that the performance of a discovery service is influenced by the distribution of classes of resources, a class of resource being a set of resources that satisfy some given constraints on resource properties, as discussed in Section 4.

(iii) In a Grid, it is feasible to identify, for each VO, a subset of powerful nodes having high availability properties; these nodes can be used as super-peers.

These considerations guided us through the design of membership and discovery services. For the sake of simplicity we suppose that only one super-peer is associated to each VO, i.e. we will not consider *redundant* super-peers. Whenever a VO wants to join the Grid, the corresponding super-peer must know the address of at least another super-peer and explores the topology of the system to constitute its *neighbour set*. A

super-peer accomplishes two main tasks: it is responsible for the communications with the other VOs, and it maintains metadata about all the nodes of the local VO. The set of nodes belonging to a VO (i.e. the super-peer and the ordinary nodes) is also referred to as a *cluster* in the following.

As shown in Figure 1, the super-peer model exploits the centralized/hierarchical information service provided by the Grid infrastructure of the local VO: e.g. the MDS-2 service of GT2 [2] or the Index Service of GT3 [3]. It is not necessary that the same Grid framework is installed in all the VOs: it is only required that the super-peers are able to communicate with each other using a standard protocol and that each super-peer knows how to interact with the information service of the local VO.

The resource discovery protocol, exploited by the discovery service, is defined as follows. Query messages generated by a Grid node are forwarded to the local super-peer. The super-peer examines the local information service to verify if the requested resources are present in some of the nodes belonging to the local VO, and in this case sends to the requesting node a queryHit containing the IDs of those nodes.

Furthermore, the super-peer forwards a copy of the query to a selected number of neighbour super-peers, which in turn contact the respective information systems and so on. Whenever a resource, matching the criteria specified in the query, is found in a remote VO, a queryHit is generated and is forwarded along the same path back to the requesting node, and a notification message is sent by the remote super-peer to the node that handles the discovered resource.

The set of neighbours to which a query is forwarded is determined through an empirical approach. Each super-peer maintains statistics on the number of queryHits received from all the known super-peers. The super-peer forwards a query to the neighbour super-peers from which the highest numbers of queryHits were received in the past. The maximum number of neighbours to which a query is forwarded can be tuned on the basis of the network configuration, as discussed in Section 4.

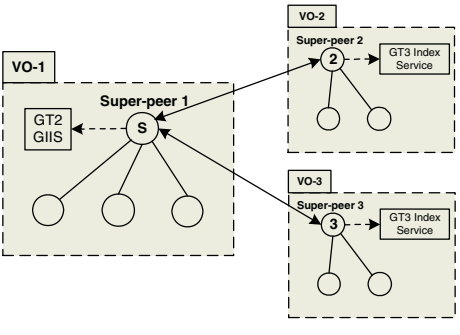


Fig. 1. A Grid network configuration exploiting the super-peer model

A number of techniques are adopted to decrease the network load. (i) The number of hops is limited by a Time-To-Live (TTL) parameter; the TTL is decremented when the query is forwarded between two super-peers, i.e. between two different VOs. (ii) Each query message contains a field used to annotate the nodes that the query traverses along its path. A super-peer does not forward a query to a neighbour super-

peer that has already received it. (iii) Each super-peer maintains a cache where it annotates the IDs of the last received query messages. A super-peer discards the queries that it has already received. (iv) Whenever a super-peer, after receiving a query, finds several resources that satisfy the query constraints in the local VO, it constructs and forwards only one queryHit message containing the IDs of the nodes that own those resources. Techniques (ii) and (iii) are used to avoid the formation of cycles in the query path: technique (ii) can *prevent* cycles only in particular cases (i.e. when a query, forwarded by a super-peer, is subsequently delivered to the same super-peer), whereas technique (iii) can *remove* cycles in all the other cases (e.g., when *two* copies of a query, sent by a super-peer A to two distinct super-peers B and C, are subsequently both delivered to the remote super-peer D).

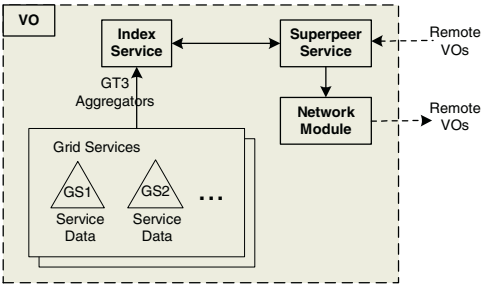


Fig. 2. Implementation of the super-peer model using the GT3 framework

```
// v = max number of neighbours
// q.list: list of hosts traversed by the query q
// q.sender: neighbour super-peer from which q has been received
// q.id: query identifier
// q.ttl: current value of ttl
For each incoming query q:
  If <q.id is in the cache> then queryInCache:=true;
  Else <put q.id in the cache>
  q.ttl -= 1;
  if ((q.ttl>0) and not queryInCache)
  {
    select at most v best neighbours
    for each selected neighbour n:
      if <n is not in q.list> {
        <Add this super-peer to q.list>
     forward a copy of q to n
      }
  }
  <ask the local information service for resources matching q>
  if <there are such resources> {
    send to q.sender a queryHit containing the IDs of the nodes owning
      the discovered resources;
    send notifications to the hosts owning the resources;
  }
```

Fig. 3. The resource discovery algorithm executed by the Superpeer Service

Figure 2 shows how the GT3 information service is used in a VO to implement the super-peer model. Such architecture extends the one presented in [11]. The Index

Service subscribes to the Service Data contained in the Grid Services published on the nodes of the local VO. Specialized GT3 aggregators periodically collect Service Data, which typically contain metadata information about Grid Services, and send it to the Index Service. The Superpeer Service is a static Grid Service that processes requests coming from the remote VOs, queries the Index Service to find resources matching the query constraints, and forwards query and queryHit messages through the Network Module. Minor modifications will be needed in this architecture to replace the GT3 framework with the WSRF-based Globus Toolkit 4 that is going to be released.

A simplified version of the resource discovery algorithm, executed by a Superpeer Service when receiving a query from an external VO, is reported in Figure 3.

4 Simulation Analysis

The performance of the resource discovery protocol, described in Section 3, was analyzed in order to assess its effectiveness in a Grid environment and estimate the influence of protocol parameters on performance indices. An event-based object-oriented simulator was used both for modelling the construction of a super-peer network, driven by the membership protocol, and for simulating the behaviour of the resource discovery protocol in very large Grid networks.

4.1 Simulation Parameters and Performance Indices

The performance of a resource discovery protocol depends on the distribution of resources among the hosts of a network. As mentioned in Section 3, in Grid systems users often need to discover resources that belong to classes of resources, rather than well defined resources. A class of resources is defined as the set of resources that satisfy some given constraints on resource properties. For example, when building a distributed data mining application [6], a user may need to discover a software that performs a clustering task on a given type of source data. Therefore the performance of a resource discovery protocol in a Grid is strictly related to the categorization of heterogeneous resources in a given application domain.

We assumed, as in [5], that the average number of elementary resources offered by a single node (peer or super-peer) remains constant as the network size increases. This average value was set to 5, and a gamma stochastic function was used to determine the number of resources owned by each node. However, as the network size increases, it becomes more and more unlikely that a new node connecting to the network provides resources belonging to a new resource class. Therefore, we assumed that the overall number of distinct resource classes offered by a network does not increase linearly with the network size. We adopted a logarithmic distribution: the number of resource classes offered by a Grid network with N nodes (where N is comprised between 10 and 10000) is equal to $5*(\log_2 N)^2$. As an example, a Grid having 1024 nodes provides 5120 resources belonging to 500 different classes.

Table 1 reports the simulation parameters and the performance indices used in our analysis. During a simulation run, a node randomly selects, with a frequency determined by the mean query generation time MQGT, a resource class, and forwards a

query for resources belonging to that class. Among the performance indices, Nres is deemed to be more important than the probability of success Psucc, since it is often argued that the *satisfaction of the query* depends on the number of results (i.e. the number of discovered resources) returned to the user that issued the query: for example, a resource discovery operation could be considered *satisfactory* only if the number of results exceeds a given threshold. The message load L should obviously be kept as low as possible. This performance index often counterbalances the success indices, in the sense that high success probabilities sometimes are only achievable at the cost of having high elaboration loads. The ratio R is an index of efficiency: if we succeed in increasing the value of R, a higher relative number of queryHit messages, containing useful results, is generated or forwarded with respect to the overall number of messages. Response times are related to the *time to satisfaction* experienced by a user: to calculate them, we assumed that the mean hop time is equal to 10 msec for an internal hop (i.e. a hop between a peer and the local super-peer) and to 50 msec for an external hop (i.e. a hop between two super-peers).

Table 1. Simulation parameters and performance indices

Parameter	Value
Network size (number of nodes) N	10 to 10000
Mean cluster size C	1 to 5000
Overall number of resources offered by the network vs. the network size	$5N$
Overall number of resource classes offered by the network vs. the network size	$5(\log_2 N)^2$
V: maximum number of neighbours to which a super-peer can forward a query	2 to 8
Time to live TTL	1 to 5
Mean query generation time MQGT	300 sec

Performance index	Definition
Probability of success Psucc	Probability that a query issued by a peer will succeed. i.e. will be followed by at least one queryHit.
Mean number of results Nres	Mean number of resources that a node discovers after its query.
Message load L	Frequency of all messages (queries, queryHits, notifications) received by a node.
QueryHits/ Messages ratio R	Number of queryHits received by a node divided by the overall number of messages received by that node.
Response times Tr, Tr(K), Tr(L)	Mean amount of time that elapses between the generation of a query and the reception of a generic queryHit (average response time), of the k^{th} queryHit, of the last queryHit.

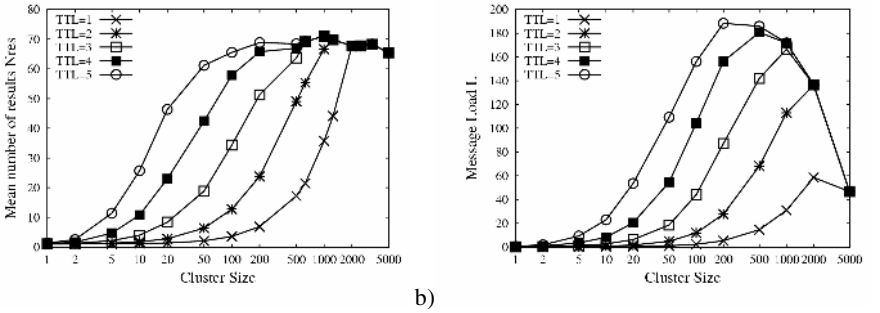
4.2 Performance Evaluation

The proposed discovery protocol was first analyzed for a Grid network having a fixed number of nodes (10000, including super-peers and simple peers), and a mean cluster size C ranging from 1 (corresponding to a fully decentralized P2P network, in which peers and super-peers coincide) to 5000 (corresponding to a network composed of two clusters). We tested different values of v and TTL, in order to analyze how those parameters can be tuned to improve performance when the mean cluster size is

known. Notice that an estimated value of the mean cluster size can be computed by exchanging information among super-peers.

Results shown in Figures 4-6 are obtained with v equal to 4. Figure 4(a) reports the mean number of discovered resources versus *C*, for TTL values ranging from 1 to 5. It appears that performance values increase with the TTL value as long as *C* is lower than 1000. Beyond this threshold, curves related to different values of TTL tend to converge. This information can be exploited when tuning the value of TTL. For example, if we want to maximize the number of results, with a value of *C* equal to 100, the TTL value should be 5 or higher, whereas if the value of *C* is higher than 500, it is almost ineffective to increase the TTL value beyond 3. The very small number of results obtained for a decentralized network, i.e. with a cluster size equal to 1, demonstrates the great advantage that comes from the use of the super-peer model. From Figure 4(b) we see that a high processing load at super-peers is a toll to pay if a high number of results are desired. Indeed the curves of message load show a trend similar to the trend observed in Figure 4(a), except for networks having very large clusters, in which a high percentage of resources is discovered within the local VO.

A trade-off should be reached between maximizing the number of results and minimizing processing load; to this aim, we calculated the R index at super-peers. From Figure 5 we see that R, for a fixed value of TTL, initially increases with *C*, as a result of the fact that the number of incoming queryHits experiences a higher increase rate than the overall number of messages. Beyond a threshold value of *C*, which depends on the value of TTL, an opposite trend is observed; the number of received queryHits falls down, due to the fact that a consistent percentage of peers are internal, i.e. situated within the local VO. Remind that a super-peer receives queryHits only from remote VOs, since it knows the resources offered by the local VO. Values of R converge to 1/3 with *C*=5000, i.e. with only two clusters in the network, for the following reason: each super-peer receives comparable numbers of internal queries (queries from local peers), external queries (queries from the other super-peer) and queryHits, because the numbers of peers in the two clusters are similar and almost each query directed to the other super-peer is followed by one queryHit message.



a) b)

Fig. 4. Mean number of results (a) and message load at super-peers (b) w.r.t. the cluster size, for different values of TTL and v=4

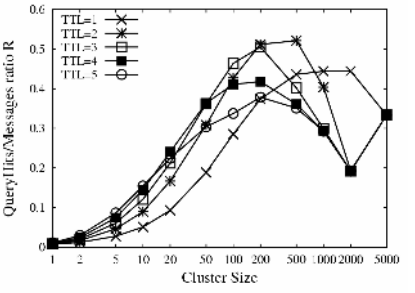
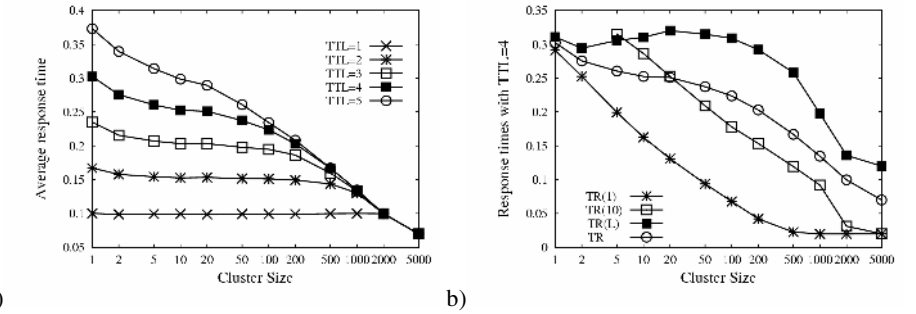


Fig. 5. QueryHits/messages ratio `R` at super-peers w.r.t. the cluster size, calculated for different values of `TTL` and `v`=4



a) b)

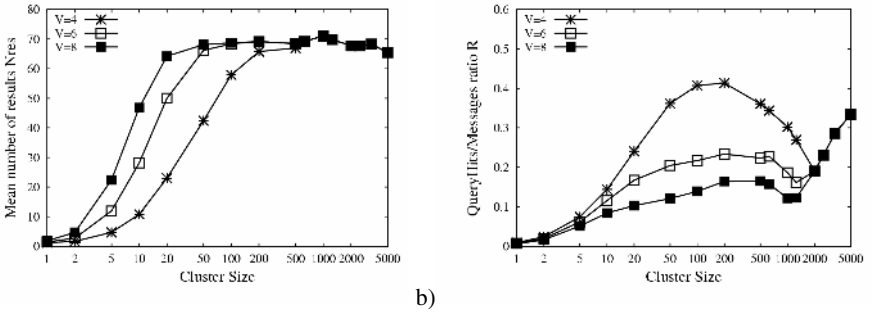
Fig. 6. Response times w.r.t. the cluster size, with `v`=4. (a): average response times for different values of `TTL`; (b): comparison between `Tr`, `Tr(1)`, `Tr(10)` and `Tr(L)`, with `TTL`=4

Figure 5 helps to identify, for a given value of C, the `TTL` value that maximizes the efficiency of the network. As the mean cluster size increases, the optimal value of `TTL` becomes smaller and smaller. For example, the optimal value of `TTL` is equal to 3 if C is 100, and to 2 if C is 1000. It is interesting to note that the highest values of `R` are obtained for cluster sizes comprised between 200 and 500 and a `TTL` value equal to 2.

Values of response times versus the cluster size are reported in Figure 6. Figure 6(a) shows that the average response time increases with the `TTL` value and decreases with the cluster size. Moreover, it is confirmed that a high value of `TTL` is advantageous only for small/medium cluster sizes. In Figure 6(b), different response time indices are compared (`TTL` is set to 4): plotted curves are related to the average response time, the response times of the first and the 10^{th} queryHit, and the response time of the last queryHit received for a given query. The values of these indices decrease as the cluster size increases, for two main reasons: (i) queries and queryHits traverse a smaller number of super-peers and (ii) a higher fraction of queryHits are internal ones, which are statistically received earlier than external queryHits. The `Tr(L)` index is an exception: it slightly increases as the cluster size increases form 2 to 100. The reason is that within that range of the cluster size the number of results

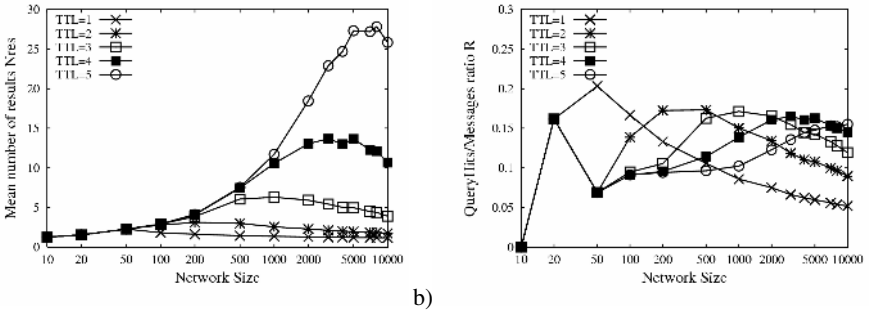
increases very rapidly, therefore there is a higher and higher probability that the last query, which is normally an external query, experiences a long response time.

Figure 7 reports the values of Nres and R obtained for a TTL value equal to 4 and a variable number of neighbours v, in order to evaluate how v can be tuned for a fixed value of TTL. The number of results significantly increases with the value of v only if the cluster size is lower than 100; with larger clusters, a value of v equal to 4 is sufficient to achieve a high number of results. Figure 7(b) shows that the values of R are maximized with v equal to 4. We can conclude that it is not convenient to set v to a value higher than 4 if the cluster size exceeds 100, because we would increase the network and processing load without increasing the number of results.



a) b)

Fig. 7. Mean number of results (a), and queryHits/messages ratio R at super-peers (b) w.r.t. the cluster size, for different values of v, and TTL=4



a) b)

Fig. 8. Mean number of results (a) and queryHits/messages ratio at super-peers (b) w.r.t. the network size, for different values of TTL, and v=4

Finally, the performance of the super-peer model was analyzed for different network sizes. The mean cluster size was set to 10, while the number of Grid nodes was varied from 10 (corresponding to a super-peer network having only one cluster) to 10000. The value of v was set to 4. It appears from Figure 8(a) that an increase in the TTL value allows for increasing the number of results only in networks having

more than 1000 nodes. Moreover, Figure 8(b) shows that the optimum TTL value, i.e. the value that maximizes R, increases with the network size. For example, in a network with 1000 nodes the maximum value of R is obtained with a TTL equal to 3, whereas in a network with 10000 nodes R is maximized with a TTL equal to 5. Thus TTL should be set to a value equal or greater than 5 only if the number of nodes exceeds 5000; for smaller networks, tuning decisions should take into account that a high value of TTL can slightly increase the number of results but surely decreases the overall efficiency.

5 Conclusions

Resource discovery in Grid environments is currently based on centralized models. Because such information systems are built to address the requirements of organizational-based Grids, they do not deal with more dynamic, large-scale distributed environments. The number of queries in such environments makes a client-server approach ineffective. Future large-scale Grid systems should implement a P2P-style decentralized resource discovery model.

The super-peer model is a novel approach that facilitates the convergence of P2P models and Grid environments, since a super-peer serves a single Virtual Organization (VO) in a Grid and at the same time connects to other super-peers to form a peer network at a higher level. This paper proposed an approach based on the super-peer model for handling membership management and resource discovery services in a Grid. In particular, a resource discovery protocol was presented and discussed. We reported simulation results obtained with different network configurations and evaluated the influence of protocol parameters (such as the number of neighbour super-peers and the time to live) on performance indices. Performance evaluation allows for efficiently tuning the values of protocol parameters when a real Grid must be deployed.

References

1. Foster, I., Kesselman, C., Nick, J., S. Tuecke, S.: Grid services for distributed system integration, IEEE Computer, 35(6) (2002) 37-46
2. The Globus Alliance: Information Services in the Globus Toolkit 2 Release, http://www.globus.org/mds/mdstechnologybrief_draft4.pdf
3. The Globus Alliance: Information Services in the Globus Toolkit 3 Release, http://www.globus.org/mds
4. The Gnutella Protocol Specification v.0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf
5. Iamnitchi, A., Foster, I., Weglarz, J., Nabrzyski, J., Schopf, J., Stroinski, M.: A Peer-to-Peer Approach to Resource Location in Grid Environments, eds. Grid Resource Management, Kluwer Publishing (2003)
6. Mastroianni, C. Talia, D., Trunfio, P.: Managing Heterogeneous Resources in Data Mining Applications on Grids Using XML-Based Metadata, Proc. of International Parallel and Distributed Processing Symposium (IPDPS), Nice, France (2003)

7. Montresor, A.: A Robust Protocol for Building Superpeer Overlay Topologies, Proc. of the International Conference on Peer-to-Peer Computing, Zurich, Switzerland (2004)
8. Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., Loser, A.: Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks, Proc. of World Wide Web Conference, Budapest, Hungary (2003) 536-543
9. Puppin, D., Moncelli, S., Baraglia, R., Tonellotto, N., Silvestri, F.: A Peer-to-peer Information Service for the Grid, Proc. of International Conference on Broadband Networks, San Jose, CA, USA (2004)
10. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications, Proc. of ACM SIGCOMM, San Diego, CA, USA (2001)
11. Talia, D., Trunfio P.: A P2P Grid Services-Based Protocol: Design and Evaluation, Proc. of the European Conference on Parallel Computing (EuroPar), LNCS 3149 (2004)
12. The Web Services Resource Framework, http://www.globus.org/wsrf/
13. Yang, B., Garcia-Molina, H.: Designing a Super-Peer Network, 19th Int'l Conf. on Data Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA (2003)

Ontology-Based Grid Index Service for Advanced Resource Discovery and Monitoring

Said Mirza Pahlevi and Isao Kojima

National Institute of Advanced Industrial Science and Technology (AIST),
Grid Technology Research Center,
Tsukuba, Ibaraki 305-8568, Japan

Abstract. This paper proposes a framework for advanced Grid resource discovery and monitoring. In the framework, a service's state data are mapped into ontologies so that service owners may enrich them with semantic and other useful data, while keeping the state data unchanged. The Index Service, based on the OGSA framework, aggregates and maintains the ontology data, and allows users to query the data by using ontology-based query languages. The Index Service also provides a Continual Query mechanism that enables users to submit ontology queries as subscription messages and to be notified when the query results change. This paper also proposes an automatic ontology update mechanism, to keep the ontology data up-to-date.

1 Introduction

The Open Grid Services Architecture (OGSA) [1] constitutes a conceptual framework for Grid computing that is based on Web services concepts and technologies. It provides standard mechanisms for discovering Grid service instances. The mechanisms define a standard representation for information about Grid service instances, denoted as *serviceData*. Globus Toolkit 3.2 (GT3.2)[1] [3] offers a Base Service, known as the *Index Service*, that provides the functionality within which serviceData can be collected, aggregated, and queried. Clients access the aggregated serviceData by using either of two mechanisms, *findServiceData* (a pull operation) or *subscription-notification* (a push operation). Clearly, the Index Service offers users considerable help with respect to resource discovery, selection, and monitoring.

The elements of serviceData (SDEs) are defined in a service's definition of the service interface, by means of an XML schema. As a result, serviceData constitutes a structured document but imposes no *semantic constraints* on the meaning of the document. Because it has no semantic constraints, serviceData

[1] GT3.2 implements the Open Grid Service Infrastructure (OGSI), which addresses detailed specifications for OGSA. Recently, the OGSI has been refactored to the WS-Resource Framework (WSRF) [2]. Since the WSRF retains, essentially, all of the OGSI concepts, the framework proposed here can easily be adapted to the WSRF.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 144–153, 2005.

cannot easily be shared across applications or domains, nor can it easily be processed by automated agents/tools. The lack of semantic constraints also makes it difficult to implement automated reasoning, which could improve resource discovery and selection. For example, suppose there are Grid Data Service Factories (GDSFs)[2] that provide access to relational databases that contain, in turn, information about computer-related books. Each factory provides access to one relational database, and each database covers one of the following topics: compilers, firewalls, and biometrics. To facilitate resource discovery, the GDSFs provide a *databaseSchema* SDE that contains logical schema of all tables existing in the database. Since the SDE contains no semantic information, it would be difficult to perform automatic resource discovery and selection. For example, it would be difficult to automatically find factories that provide access to tables that have a column containing a specific domain value, such as a book's title. It would be difficult to find factories that provide access to databases containing books that address a specific topic such as biometrics, or a more general topic such as computer security, a topic that is related to both biometrics and to firewalls.

In order to enrich serviceData with semantic constraints, one could directly modify the SDE definition in the service specification. However, that would not constitute a suitable approach because of the following. First, the modification would affect all clients that refer to the serviceData. Second, adding "nonstandard" elements to the SDE definition would require communicating the new elements to clients and, currently, there is no efficient way to publish the semantics of new elements to clients. Third, the user may not have the access privileges to update the service specification or to recompile and/or redeploy the service.

In GT3.2, the subscription-notification can be performed only for the entire SDE. An Index Service client cannot specify a SDE's part, or element of interest, in order to receive notification only when that part changes. The client is forced to accept notification messages pertaining to the entire SDE, regardless of whether the monitored value, which may be a small part of the SDE, has changed.

This paper proposes a framework for creating, maintaining, querying, and monitoring *semantic serviceData/SDEs*. Some domain-specific ontologies are defined, and SDE values of services are mapped onto the property values of the ontologies. In the framework, each service has a *Service Data Provider* (SDP), which stores the service's semantic serviceData in an SDE. The Index Service aggregates the semantic serviceData by subscribing to the SDE and storing the serviceData in an ontology repository. To keep the semantic serviceData up-to-date, an automatic ontology update mechanism is proposed. The mechanism makes use of the subscription-notification mechanism and stores access paths to the original SDE values in the semantic serviceData. Index Service clients formulate queries with an ontology-based query language for information discovery and exploration of the serviceData's semantic parts.

[2] GDSFs are the part of the OGSA-DAI software [4] that implements the Grid Data Service Specification [5].

The Index Service also provides a *Continual Query* (CQ) mechanism for advanced resource monitoring. A CQ is a standing query that monitors the update of interest and returns results whenever the update has satisfied a specified condition. With the mechanism, a client can send ontology queries as subscription messages and receive (new) query results whenever monitored values have changed.

The rest of the paper is organized as follows. Section 2 briefly describes some semantic web technologies. Section 3 describes related work. Our proposed framework and its implementation are presented in Section 4. We conclude our paper in Section 5.

2 Resource Description Framework (RDF) and Ontology

The Resource Description Framework (RDF) [6] is a W3C Recommendation that is particularly intended for representing metadata about Web resources. RDF statements, also called *triples*, consist of three parts: *subject*, which identifies the item the statement is about; *predicate*, which identifies the property of the item that the statement specifies; and *object*, which identifies the value of that property. The RDF Schema (RDFS) extends the RDF standard by providing the means to specify vocabularies/terms used in RDF statements. To do this, the RDFS pre-specifies a body of terminology, such as `Class`, `subClassOf`, and `Property`, which forms a basis for building a hierarchical structure.

RDF and RDFS standards are widely used to build ontologies. An ontology (or '*shared understanding*') consists of explicit formal specifications of terms in the domain, and the relations between them. An ontology allows people or software agents to share a common understanding of the structure of information.

For effective storage and query of ontologies, use of a high level query language is essential. The numerous existing ontology repositories include Jena [7] and Sesame [8]. For querying ontologies, the repositories support query languages such as RDQL [9] and SeRQL [10].

3 Related Work

Ontology-based Matchmaker (OMM) [11] is an ontology-based resource selector for solving resource matching in the Grid. It consists of three components: *ontologies*, used for expressing resource advertisements and job requests; *domain background knowledge*, which captures additional knowledge about the domain; and *matchmaking rules*, which define the matching constraints based on the ontologies and background knowledge. In OMM, resource providers periodically advertise their resources to a matchmaker by sending advertisement messages, based on a resource ontology. Requesters formulate a job request to the matchmaker, basing it on a request ontology. On receiving a job request, the matchmaker activates the matching rules to find a list of potential matches.

Our work differs from the OMM, in the following aspects. First, the OMM requires resource providers to construct advertisements and to send them, peri-

odically, to the matchmaker, whereas our system uses an automatic service state mapping and ontology update operation. Second, OMM uses different ontologies for resource advertisements and job requests, while ours does not. Using different ontologies for the two kinds of processes results in the need to define rules that match advertisements and job requests. As a result, updating ontology vocabularies results in a modification of the matching rules. In addition, the OMM does not allow easy use of other rule-based engines, because the matching rules and background knowledge must be transformed into the engine's rules. By contrast, our system can easily use the existing ontology repositories and explore the power of given ontology-based query languages. Third, the OMM lacks a CQ mechanism, so the requestor cannot easily and effectively monitor the resource advertisements.

Several approaches have already been suggested for adding semantics to the Web service standard for improved service discovery. METEOR-S [12] is a framework for semi-automatically marking up Web service descriptions with ontologies. It provides algorithms to match and annotate WSDL files with relevant ontologies. By contrast, UDDI-M^T [13] deals with Web service directories. It enables users to specify metadata for the information stored in the directories. The metadata are stored (locally) in a Jena repository and can be queried using RDQL. UDDIe [14] also adds metadata to the directories data but with a service leasing mechanism. These systems do not deal with the service state of Grid services but, rather, with Web service standards.

Continual Query for Web scale applications has been studied in the literature. The goal is to transform a passive web into an active environment. OpenCQ [15] allows users to specify the information they want to monitor by using an SQL-like expression with a trigger condition. Whenever the information of interest becomes available, the system immediately delivers it to the users. The rest of the time, the system continually monitors the arrival of the desired information and pushes it to the users as it meets a threshold. NiagaraCQ [16] goes further, by grouping CQs that have similar structures so they can share common computation. Our CQ mechanism is different in that it monitors ontology instances/data[3]. Hence, our system uses RDQL to formulate CQs. Furthermore, the mechanism is adapted to the Grid environment because it uses the OGSA subscription-notification mechanism to deliver the CQ execution results.

4 Proposed Framework

4.1 General and Service State Ontologies

To add semantic information to serviceData, we map the SDE values into ontology property values. Ontologies used in this framework can be categorized

[3] The term 'ontology instance' corresponds to the RDF (facts), while the term 'ontology' (without instance) corresponds to the RDF schema.

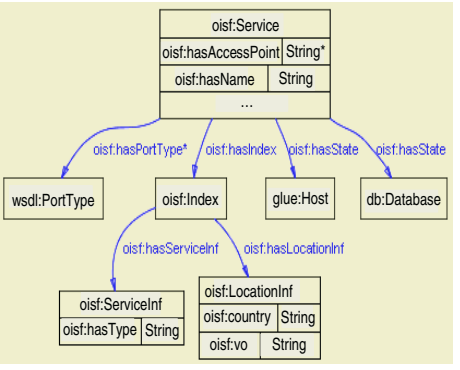


Fig. 1. General Ontology

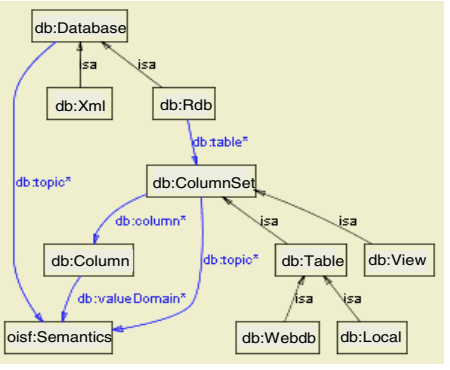


Fig. 2. Database Service Ontology

into two types: *General Ontology* and *Service State Ontology*. The former is for service-general information, and the latter is for service-specific information.

Fig. 1 shows part of the General Ontology. The part shown defines certain classes and properties as including general service information, such as service location (country and VO names), service types (e.g., computing and database service), service portType, service access point(s), and service name. Note that the *oisf:hasState* property refers to Service State Ontology instances. Currently, we define two Service State Ontologies: Database Service Ontology (shown in Fig. 2) and Computing Service Ontology. The Database Service Ontology is based on a database logical schema defined by the CIM-based Grid Schema Working Group of GGF [17]. The Computing Service Ontology is based on the GLUE schema defined by the EU-DataTAG and US-iVDGL projects [18].

4.2 SDE Value Mapping and Semantic ServiceData Creation

SDE values (in serviceData) are mapped into property values of a Service State Ontology. To this end, each SDE value is associated with a specific class in the ontology, called the `SDEInfo` class. This class has the following properties.

- *oisf:value*, which stores the SDE value.
- *oisf:valueState*, which stores the characteristics of the SDE value ('*static*' or '*dynamic*'). This property is used to efficiently update the *oisf:value* property value when the corresponding SDE value changes (see Section 4.3).
- *oisf:definedIn*, which refers to a *wsdl:portType* class instance.
- *oisf:hasSDEMapping*, which refers to an `SDEMapping` class instance. This class instance stores the access path information of the SDE value, such as the SDE name, the XPath expression to retrieve the value from the SDE, and the XML namespace mapping used in the XPath expression.

Fig. 3 shows the instances of `db:Rdb`, `db:Driver` (i.e., `SDEInfo` class), and `oisf:SDEMapping` classes defined in the Database Service Ontology. `db:Driver_`

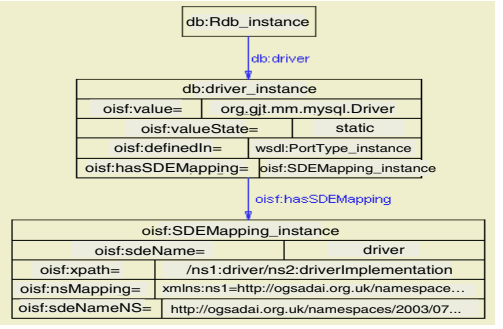


Fig. 3. SDE mapping

Algorithm: Maintain *semanticData* SDE.
Input: Ontology instances (semantic serviceData), *inst*.
Method:
1: Put *inst* into *semanticData* SDE and publish the SDE to Index Service;
2: Extract SDE names contained in Service State Ontology instances ($\in$ *inst*);
3: Subscribe to the extracted SDEs;
4: On receiving an SDE update message, *mes*,
5: $sName \leftarrow$ extract SDE name from *mes*;
6: Get `SDEInfo` instance ($\in$ *inst*) where *value*(`SDEInfo`.*oisf:valueState*) ='dynamic';
7: For each obtained `SDEInfo` instance, `sInfo`,
8: `sMapping` $\leftarrow$ get its `SDEMapping` instance ($\in$ *inst*);
9: If *value*(`sMapping`.*oisf:sdeName*) = *sName*,
10: *xpath* $\leftarrow$ get XPath expression from `sMapping`;
11: *val* $\leftarrow$ apply *xpath* to *mes*;
12: `sInfo`.*oisf:value* $\leftarrow$ *val*;

Fig. 4. *semanticData* SDE maintenance algorithm

`instance` corresponds to the *driver* element content in the serviceData of a database service (i.e., GDSF). It stores a driver implementation value in the *oisf:value* property. The XPath expression in `oisf:SDEMapping_instance` specifies how to retrieve the value from the *driver* element.

Ontology instances (semantic serviceData) of a service are created as follows. General Ontology instances are created manually/semi-automatically by a resource/service owner, using ontology editors such as Protëgë [19]. Service State Ontology instances are created, largely automatically, by the use of a simple parser. The parser reads a mapping file that maps each SDE value to a corresponding `SDEInfo` class in the ontology.

4.3 Service Data Provider (SDP)

To hold ontology instances, each service instance is associated with a Service Data Provider (SDP). An SDP is a transient service that has a special SDE, known as *semanticData*, to store semantic serviceData of the represented service.

Fig. 4 shows an algorithm used by an SDP to automatically maintain its *semanticData* SDE. Note that *value*(`inst`.*oisf:prop*) denotes the value of property *oisf:prop* of class instance `inst`. On receiving ontology instances, an SDP stores the instances into the *semanticData* SDE and publishes the SDE to Index Service (line 1). Publishing the SDE to Index Service makes the SDE available to be queried by clients. The SDP then extracts all distinct SDE names contained in the Service State Ontology instances (line 2). This is accomplished by getting *oisf:sdeName* property values from `SDEMapping` class instances. Next, the SDP sends subscription messages to the (represented) service in order to subscribe to the extracted SDEs (line 3). When the service updates an SDE value, the SDP receives a notification message containing the updated SDE (line 4). The SDP then extracts the SDE name from the message (line 5).

Next, the SDP updates the stored ontology instances (lines 6–12). To do this, it first gets all `SDEInfo` class instances for which the *oisf:valueState* property values are 'dynamic' (line 6). Then, for each obtained `SDEInfo` instance, the SDP gets the `SDEMapping` instance referred by the `SDEInfo` class instance (i.e., *value*(`SDEInfo`.*oisf:hasSDEMapping*)) (line 8). If the `SDEMapping` instance corresponds to the SDE contained in the notification message, then the SDP gets the XPath expression from the mapping instance (line 10), applies the expression to the notification message (line 11), and finally puts the obtained result/value[4] into the *oisf:value* property of the `SDEInfo` instance (line 12).

An SDP will be destroyed when the represented service stops. Before the SDP is destroyed, an unpublish message is sent to the Index Service to remove the service's semantic serviceData from the Index Service's repository.

4.4 Ontology-Based Grid Index Service (Ont-GIS)

Repository Maintenance. The Index Service (Ont-GIS) has an ontology repository, which it allows clients to query by using an ontology-based query language. Queries are formulated based on the General and Service State Ontologies. We use Jena [7] as the ontology repository and RDQL [9] as the query language.

Fig. 5 shows the Ont-GIS architecture. Ont-GIS collects *semanticData* SDEs from several SDPs and stores their contents in the repository. The content of each *semanticData* SDE is stored as a Jena persistent model[5]. On receiving a publish request from an SDP, the Subscription Manager pulls the *semanticData* SDE from the SDP and stores the content by passing it to the Update Manager. The Subscription Manager then subscribes to the *semanticData* SDE and waits for a notification change message.

On receiving a notification change message, the Subscription Manager passes *update data* to the Update Manager and CQ Manager. Based on the update data,

[4] Since the XPath expression always points to a specific SDE value, the result of its application is always singular.

[5] A Jena Model is a (Java) class instance that provides operations to manipulate ontology instances stored in the model. A model backed by a database engine is called a persistent model.

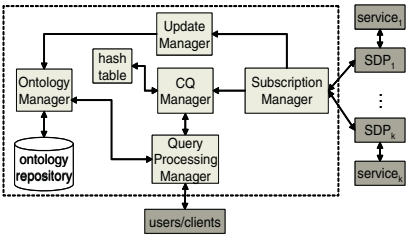


Fig. 5. Ont-GIS Architecture

the Update Manager performs an ontology update operation with the assistance of the Ontology Manager. The CQ Manager executes CQs that are related to the update data. Update data consists of an (RDF) resource name, a model ID, and the updated value. The resource name identifies a resource (i.e., `SDEInfo` class instance) for which the *oisf:value* property value has changed, and the model ID identifies the model in which the resource is stored.

Query Execution. An RDQL query is executed in a simple manner. On accepting a query, the Query Processing Manager executes the query by the help of the Ontology Manager and returns the execution results to the users/clients.

On the other hand, execution of a CQ requires two steps. Fig. 6 shows the algorithms used by the CQ Manager. The CQ Manager has a *hash table* to store CQs and query execution information. Each location in the hash table is a *set* so that it can store more than one value. $InsertH(k, val)$ is a hash function that inserts value val, based on key k. $LookupH(k)$ retrieves an entry (a set) from the hash table based on key k.

The query subscription process (left algorithm) constitutes the first step in CQ execution. A client sends a query subscription request to the Query Processing Manager, which, in turn, passes the request to the CQ Manager. The CQ Manager then creates a *unique SDE* in Ont-GIS's serviceData (line 1), executes the CQ (which is included in the request) with the help of the Query Process-

Algorithm: Register a CQ.
Input: A CQ, cq.
Output: An SDE name.
Method:
1: Create a unique SDE, $uSDE$;
2: $res \leftarrow$ execute cq;
3: $mSet \leftarrow$ IDs of models that match cq;
4: For each model ID $m \in mSet$,
5: $infSet \leftarrow$ get monitored resources of m from res;
6: $InsertH(m, \langle cq, infSet, uSDE \rangle)$;
7: Return $uSDE$;

Algorithm: Execute a CQ.
Input: resource rn and model ID m.
Method:
1: $tSet \leftarrow LookupH(m)$;
2: If $tSet$ is not empty,
3: $mtSet \leftarrow$ all tuple from $tSet$ s.t. the monitored resource set contains rn;
4: For each tuple $t \in mtSet$,
5: $res \leftarrow$ execute CQ contained in t;
6: Put res into SDE mentioned in t;

Fig. 6. CQ registration and execution algorithm

ing Manager (line 2), and gets Jena model IDs from the execution results (line 3). Next, the CQ Manager updates its hash table. For each matched model ID, it inserts a tuple into the hash table, with the model ID as a key (lines 4–6). The tuple consists of the given CQ, the monitored resource name set extracted from the CQ execution results for the model, and the created unique SDE name. A monitored resource is a resource (i.e., `SDEInfo` class instance) for which the *oisf:valueState* property value is 'dynamic'[6]. Finally, the CQ Manager returns the unique SDE name to the client (line 7). A client receiving the SDE name will (immediately) subscribe to the SDE to receive the CQ execution results.

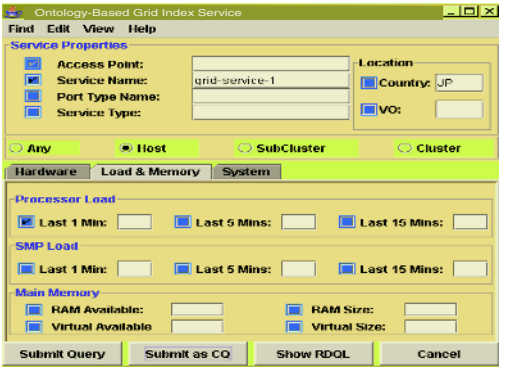


Fig. 7. A GUI for clients

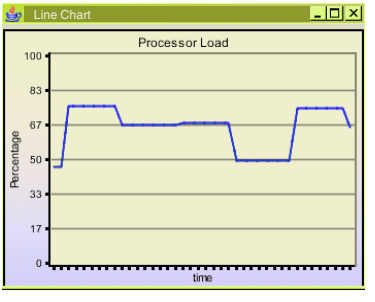


Fig. 8. Line chart of processor load

The second step is that of the CQ query execution process (right algorithm). When the CQ Manager receives the *updateData* (i.e., resource name and model ID) from the Subscription Manager, it uses its hash table to search for CQs that monitor the changes of the given model (line 1). If it finds any, the CQ Manager then gets all tuples, where the monitored resource set contains the given resource name (line 3). Finally, the CQ Manager executes all CQs contained in the tuples and puts the execution results into the corresponding SDEs (lines 4–5). Since clients whose CQs are executed have subscribed to the SDEs, they receive the execution results through notification messages sent by Ont-GIS. It is important to note that because the execution results are delivered to the clients through the OGSA subscription-notification mechanism, Ont-GIS can also collaborate with the GT3.2's Index Service.

4.5 Implementation

We have implemented the proposed framework using Java. Currently, Ont-GIS is deployed in GT3.2's service container and is able to collect semantic serviceData from computing and database services. Fig. 7 shows a graphical user interface (GUI) that helps clients construct an RDQL query. The upper and lower search

[6] Since CQ always deals with dynamic SDE values, we assume CQ contains triple (?r, *oisf:valueState*, 'dynamic'), where ?r is a variable.

forms of the GUI correspond to the General and Computing Service Ontology, respectively. The GUI also allows clients to submit an RDQL query as either a pull or a push query (CQ). Fig. 8 shows a line chart of the execution of a CQ, which monitors the last-1-minute processor load of a resource of the type `Host`.

5 Conclusions and Future Work

Two clear benefits of enriching serviceData with semantic information are those of automatic resource discovery and improved search results. Besides allowing users to explore the semantic parts of the serviceData, Ont-GIS also provides the CQ mechanism for efficient and effective resource monitoring. This mechanism uses the OGSA subscription-notification mechanism that enables it to cooperate with the GT3.2's Index Service. We are now considering a distributed architecture for Ont-GIS. This reason for this is that the Grid is distributed by nature and, as such, always deals with a great number of resources.

References

1. Foster, I., et al.: The physiology of the grid: An open grid services architecture for distributed systems integration. Globus Project (2002)
2. http://www.globus.org/wsrf/.
3. http://www-unix.globus.org/toolkit/.
4. http://www.ogsadai.org.uk.
5. http://www.cs.man.ac.uk/grid-db/documents.html.
6. http://www.w3.org/RDF/.
7. http://jena.sourceforge.net/.
8. Broekstra, J., et al.: Sesame: A generic architecture for storing and querying RDF and RDF schema. In: Proc. of ISWC 2002. Volume 2342. (2003) 54–68
9. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.
10. http://www.openrdf.org/doc/users/ch05.html.
11. Tangmunarunkit, H., et al.: Ontology-based resource matching in the grid—the grid meets the semantic web. In: Proc. of SemPG03. Volume 2870. (2003) 706–721
12. Patil, A., et al.: METEOR-S web service annotation framework. In: Proc. of the World Wide Web Conference. (2004) 553–562
13. Miles, S., et al.: Personalised grid service discovery. In: Proc. of 19th Annual UK Performance Engineering Workshop. (2003) 131–140
14. ShaikhAli, A., et al.: UDDIe: An extended registry for web services. In: Proc. of Workshop on Service Oriented Computing: Models, Architectures and Applications. (2003)
15. Liu, L., et al.: Continual queries for internet scale event-driven information delivery. Knowledge and Data Engineering **11** (1999) 610–628
16. Chen, J., et al.: NiagaraCQ: a scalable continuous query system for Internet databases. In: Proc. of the ACM SIGMOD Conf. (2000) 379–390
17. https://forge.gridforum.org/projects/cgs-wg.
18. http://www.cnaf.infn.it/ sergio/datatag/glue/.
19. http://protege.stanford.edu.

Grid Service Based Collaboration for VL-e: Requirements, Analysis and Design

A. de Ridder, A.S.Z. Belloum, and L.O. Hertzberger

Faculty of Science, IvI, University of Amstedam Kruislaan 403,
1098SJ Amsterdam, The Netherlands
{adridder, adam, bob}@science.uva.nl
http://www.vl-e.nl

Abstract. The Virtual Laboratory for e-Science seeks to provide users with a collaborative environment in which they will be able to work together across time and space while using Grid technology. In this paper we will define the requirements for collaboration in the VL-e. An in depth study of the Userlist, Instant Messenger and Telepointer has been done and a Grid Service based architecture has been designed for the Userlist and Instant Messenger as a first step for the VL-e collaborative System. The architecture consists of three Grid Services and a client. The Services have been built using the Globus Toolkit [5] in combination with the Java Shared Data Toolkit [3].

1 Introduction

One of the most interesting developments in computer science must be Grid technology. By allowing users to share resources on a global scale it promises virtually unlimited processor power, infinite storage, sharing of large databases, sharing of expensive equipment, etc.

Grid technology is available through toolkits that provide the user with a collection of low-level services. These can be used to develop Grid applications. To harness the strength of Grid technology for a variety of applications and to make the Grid available to a larger public, the Grid-based Virtual Laboratory of AMsterdam (VLAM-G) [1] and its follow up the Virtual Laboratory for e-Science (VL-e) [12] seek to provide a layer between the low-level Grid and the application layer. The Virtual Laboratory must be capable of handling large datasets coming from experimental devices regardless of the place and time of actual readings [12].

The VL-e seeks to provide global collaboration not only by global sharing of resources, but also by providing a shared-workspace environment that will allow users to collaborate in real-time. Not only will such an application have to deal with distributed computing issues, such as concurrency control, it will also have to deal with issues specifically related to collaboration, such as awareness, GUI consistency, communication, etc.

The VL-e project will extend the features of its predecessor, the VLAM-G, where possible. However, since the VL-e is a grid-based project, its architec-

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 154–163, 2005.

ture must follow the development in this domain. Since the grid is shifting to a more service-oriented approach, the VL-e architecture is being redesigned to fit this paradigm [9]. Using a service-oriented approach has some important consequences. For one, individual components can be addressed only via their published interface, which is made available to the network. This allows for decoupling between individual components, as the only connections between them is via these interfaces. Furthermore services and multiple instances of a single service can be located on different resources. When a resource fails, the same service may still be available on another resource. This gives greater overall fault tolerance. To achieve location transparency, applications search for services in a directory after which they connect to them dynamically at run-time. This allows for code mobility, as the client does not care where the service is located [13].

Grid technology allows for a special kind of services: Grid Services. Grid Services are basically Web Services with enhanced capabilities, including notifications (events between client and server), lifecycle management, and statefulness. The latter allows storage of information on the server side. Users can access a Grid Service simultaneously and work on the same data.

In this paper we develop part of the VL-e's collaborative environment. Section 2 gives a general introduction to collaboration. The requirements of the Collaborative System are defined in Section 3. In Section 4 the designed architecture is discussed and results are shown in Section 5. Finally a conclusion is drawn and some light is shed on future work in Section 6.

2 Collaboration in General

Collaboration is about two or more people working together towards a joint goal. It differs from cooperation in that with collaboration all sides benefit from working together and that it is on an equal basis. Cooperation on the other hand can also benefit only one side and is often more connected to hierarchy. Working together towards a joint goal requires means to achieve this goal, for example tools to perform an experiment, as well as means that make it possible to work together, such as providing ways that allow users to communicate.

An important aspect of collaboration is communication. When two people are engaged in verbal communication they exchange words, but much more is involved: intonation, body-language, body odor, etc [6]. These non-verbal elements add information about a person's character, his mood and provide better insight on how to proceed with the discussion. When face-to-face communication is not possible, technologies are available which allow people to collaborate over distances. One of the best known and widely used technologies is teleconferencing, where two or more people meet via the telephone. This technology provides intonation but removes any physical communication, thereby reducing its effectiveness.

Collaborative systems tend to fail, or at least fall short far of expectations. There are many reasons for this, including lack of common goals, social differ-

ences between collaborators, difficulty of testing groupware and organizational culture [6].

Grudin [7] points out that in order for an application to succeed it has to be beneficial to all its users. The VL-e allows users to make use of the power of the Grid. By making it a collaborative system users will be able to perform experiments together, discuss results, call in expertise from others, etc. This makes it beneficial to all users.

Gutwin and Greenberg [6] believe that problems with groupware are often caused by a failure to support basic necessities. They introduced the Mechanics of Collaboration, "the things groups have to do in shared workspaces, over and above what an individual has to do, in order to carry out a task" [6]. They have combined these Mechanics with Nielsen's Usability Heuristics to create heuristics for groupware [2]. These are meant for validating the quality of software but can also be used to aid in defining the requirements of a collaborative system.

3 VL-e Requirements for Collaboration

As stated in the Introduction the VL-e will be a shared workspace environment. The Mechanics of Collaboration, referred to in Section 2, are therefore valid for the VL-e project. Furthermore, actions in shared-workspaces are limited to few types [6] and these types will have to be supported. Based on these as well as on other sources (e.g. [11]), the VL-e's collaborative requirements can be defined. They have been split into groups, beginning with the ones crucial for the collaborative environment and ending with optional enhancements.

3.1 Basic Collaborative Requirements

The first set of basic collaborative requirements should allow for minimal collaboration in a single workspace. Minimal collaboration should allow users to join a session, navigate through the shared workspace, explore artifacts[1], find other users and establish contact with them, and communicate with them, through gestures as well as verbs. The most basic support for finding users and communicating with them, are a userlist, a Telepointer and an Instant Messenger.

The second set of basic collaborative requirements should provide the means to allow for creation and alteration of artifacts in a collaborative session. As now artifacts can be changed while collaborating, there is need for data consistency management. With artifacts being shared and altered, access control against unwanted alteration must be provided for the artifact's owner. Additionally, awareness is crucial for the success of a shared workspace environment as users need to be aware of other users' actions. By providing a radar view of the workspace, also showing the pointers of all users, such awareness can be provided. Finally, with multiple users altering the project simultaneously, there is now need for fault tolerance in case of an abnormal termination.

[1] In a shared workspace this refers to any component located on the workspace.

3.2 Advanced Collaboration Requirements

Additional protective features can be added, such as undo/redo and merging. Asynchronous communication will allow communication across time and has the additional benefit of storing previous communication. Feedback and feedthrough of artifacts provides additional awareness, for example by providing information on how an artifact evolved. Adding audio as a communicative device can be an enormous improvement as it allows users to use their natural language. This is in many cases is far more efficient than having to type a message and allows users to better convey emotion. Allowing for the construction of larger objects from component pieces not only reduces the chaos on the workspace, but may also make the entire component reusable. Furthermore the workspace should allow for the management of an autonomous system represented in the workspace. The potential of video as a collaborative device is still uncertain. As Sellen [4] suggested, video does not appear to be an improvement compared to high quality audio and therefore should be considered ultimately.

3.3 Optional Collaboration Requirements

The Mechanics of Collaboration suggest additionally adding means for intentional and appropriate gestural communication, and providing consequential communication of an individual's embodiment. Even though these features may help collaboration, it is questionable whether it is realistic to propose them as a requirement for the VL-e.

4 Architecture Design

Besides the collaborative requirements discussed in Section 3, the VL-e has additional requirements, which are not related to collaboration but which influence the design of the collaborative system. As mentioned in Section 1, the VL-e is being redesigned to have a service-oriented architecture. The collaborative system must therefore be service-oriented as well. The collaborative system benefits from low-level services offered by the Grid, such as security, as well as from the possibility to create statefull Grid Services.

For the VL-e the client has to be light-weight, so most of the processing must be done at the server side. The VL-e does not only promote collaboration within a single VL-e session[2] but also cross-session collaboration; multiple VL-e sessions should be able to communicate allowing a global collaboration within VL-e. This opens up a new range of possibilities, such as communication outside of a VL-e session, inviting someone to join, etc. Security is a must in the VL-e environment, only people with access to a session are allowed to communicate over a dedicated channel.

[2] In the context of the VL-e a session refers to an active project.

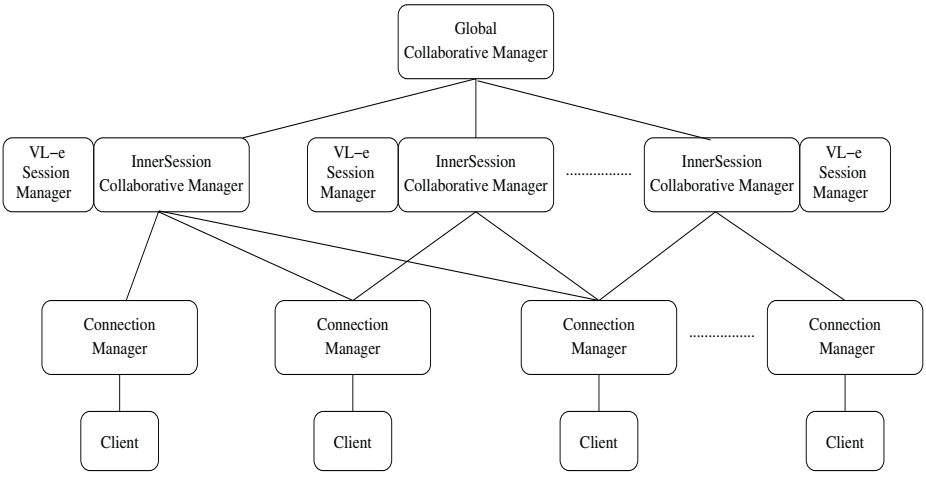


Fig. 1. Architecture Functional Overview

Use cases were developed and analyzed, available in [10], and a (Grid Service based) Userlist and Instant Messenger have been develeped, as a first step for the collaborative system. The architecture has three Grid Services and a client (Figure 1). Each client is connect to one Connection Manager (CM), which provides the functionality for the light-weight client. CMs can be connected to one or more Inner-Session Collaborative Managers (ISM). An ISM provides the collaborative environment for a VL-e session and is created by that session's VL-e Session Manager. All ISMs are connected to the Global Collaborative Manager (GM). Only one GM exists per VL-e deployment. We will explain each component in more detail in the next subsection.

4.1 Client-Side

The client's functionality is limited, as we are providing a light-weight client. The architecture is shown in Figure 2. If a user wants to join a VL-e session, his UserManager contacts the session's ISM on his behalf. After the user has joined, he can communicate with other users via the GUIs. These send messages to the CommunicationsHandler component which communicates with the user's CM. The CommunicationsHandler also receives incoming messages from the CM, which it forwards to the appropriate Handler. This component then ensures that the GUI displays the received message.

4.2 Connection Manager

The Connection Manager is a Grid Service which allows for the light-weight client. Each client communicates with his CM, which acts on his behalf on the

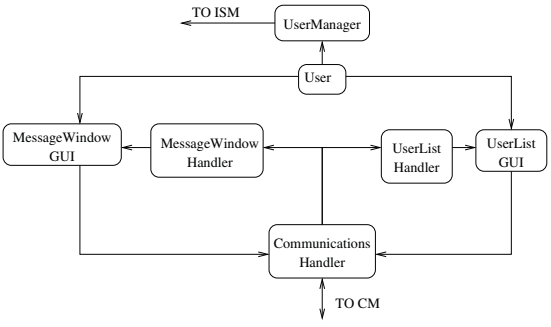


Fig. 2. Client-Side Architecture

server side. The CM joins ISMs on behalf of the client. The architecture is shown in Figure 3.

When a message is to be sent, the client side sends it to the CM. If the message was created by the client's userlist, the CM forwards the message to the UserList component. If it was created by a message window it is sent to the MessageWindow component. Here, either a TextMessage or a UserListMessage, which contains the message and necessary information retrieved from the MessageWindow Administration component, is created. This message is then sent to the MessageHandler component which then forwards it to the MessageSender.

Messages are transported between CMs over channels. The MessageReceiver component is responsible for receiving them. A received message is sent to the MessageHandler component, which decides upon the appropriate action. If the message is a text message, the MessageHandler will forward the message to the MessageWindow component and if it is a userlist message it will be forwarded to the UserList. The MessageWindow and UserList use the MessageWindow Administration component to retrieve information concerning existing conversations. When the necessary information has been retrieved, the message and the information is placed in a buffer, until it is retrieved by the CM. The CM then sends it to the client.

4.3 Global Collaborative Manager

The Global Collaborative Manager (GM) is a Grid Service that connects the individual sessions, allowing users from separate session to communicate with each other. Its architecture can be seen in Figure 4.

The GM has a MessageSender and MessageReceiver component, which allow him to communicate over channels. The channels created by the GM are joined by ISMs, thus connecting the individual ISMs.

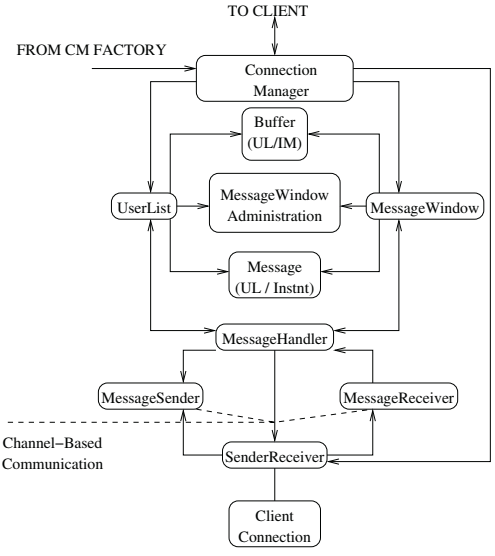


Fig. 3. Connection Manager Architecture

4.4 Inner-Session Collaborative Manager

The third Grid Service, the Inner-Session Collaborative Manager, provides each VL-e session with its own collaborative environment. Users join channels made available by the ISM. An ISM can forward a message to another ISM, via the channel made available by the GM, if a message is sent outside a session. As it is to be expected that nearly all communications take place inside a session and very little outside a session, network congestion should be less than with a architecture where all CMs communicate via the GM. Also, if the GM fails, users are only unable to send messages outside their sessions, making the ISM-based architecture more fault-tolerant. The architecture of the ISM is shown in Figure 5.

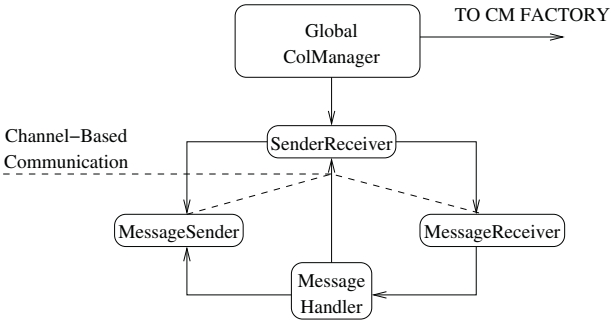


Fig. 4. Global Collaborative Manager Architecture

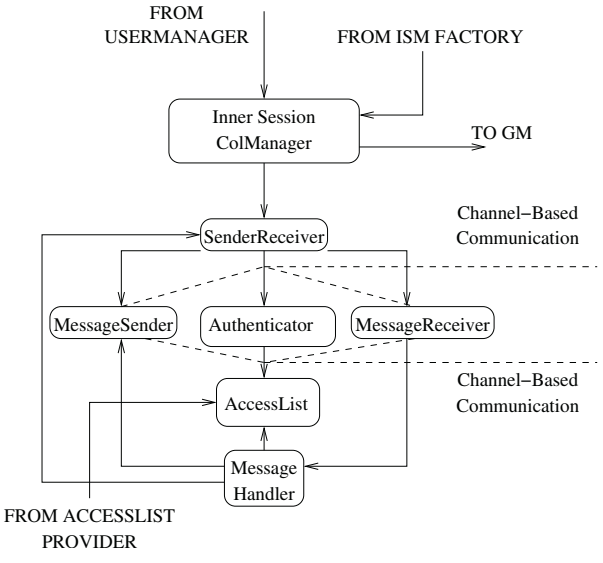


Fig. 5. Inner-Session Collaborative Manager Architecture

When a client wants to join the ISM, the ISM requests the SenderReceiver component to invite the client's ConnectinClient to join the session. The SenderReceiver then invites the client after authenticating him. The Authenticator component uses the AccessList component to validate that the client is allowed to enter the VL-e session. The MessageSender and MessageReceiver components of the ISM can send and receive messages over and from the channel created by the GM as well as over and from the channel of the ISM.

5 Results

For the implementation of the VL-e collaboration component we made use of a collaborative toolkit. Many of the groupware toolkits available on the Internet are unsuitable for the VL-e project. To determine suitability of the toolkits, we used the following selection criteria: customizable architecture, Java, recent update, usable for a wide range of applications, small customizable components. For more information on groupware toolkits we refer to [10].

We selected the Java Shared Data Toolkit [3] to aid in our work as it fits all the requirements. Instead of providing entire components, JSDT provides small building blocks to create a component, allowing it to be used to develop a wide range of applications and making it highly customizable. It is written in Java and is still being improved.

As most functionality is offered by the CM, most of the implementation been done there. Nearly all of the components at the CM side have been implemented to some extent. The ISM and GM are less complete. Not all of their components have been implemented, as not all were required yet.

The Collaborative System currently allows the creation of ISMs and CMs, users (and therefore CMs) to login into one or more ISMs, message sending between users, and changing availability in the userlist. Authentication, the notification system to invite another user to join a session and communication outside of a session are presently not fully implemented. The mechanisms for authentication have been included, but the component still has to be extended by using the VL-e information system component[8]. For communication outside a session we will use many of the mechanisms already used for communication inside a session, but it will also require mechanisms that send a message from one ISM to another via the GM. The invitation mechanism is mostly functional, but requires communication outside a session. The Collaborative System also currently lacks support for defining a personal userlist.

During the implementation several problems were encountered and one of them is worth mentioning since it is related to the current Grid Service architecture OGSI. The problem that occurred was that messages would arrive faster at the CM than that it could deliver them to the user. The Grid Service notification system notifies a client that some data has changed, after which the client retrieves this data. This is slower than the channel based JSDT communications. To solve this problem we opted for a simple solution using buffers.

6 Conclusion and Future Work

We defined a set of collaborative requirements for the VL-e, which is aiming at providing a shared workspace environment. We included basic necessities in the requirements and grouped our requirements by priority. As a first set to be developed we believe it is necessary to provide: session control, allow users to explore the workspace, a Userlist, an Instant Messenger, and a Telepointer. As a first step we developed a Grid Service based architecture for a Userlist and an Instant Messenger.

Grid Services improved our work in several ways. All three Grid Services make use of the statefullness by storing information at the server side. For the GM persistence is crucial, as there is only one GM per VL-e deployment. The Connection Manager benefited from being a Grid Service by using notifications. Furthermore, instance of both the ISM and CM can be created by a call to their factories, making it is easy to expand the system to allow for many sessions and many users. Also, it is possible to use multiple factories of the ISM and CM, creating location transparency and a more fault tolerant system; if one of the factories fails, users can switch to another. So far the only problem encountered is that a notification-based system may be too slow.

At this point our design and implementation only provides basic collaborative means. The VL-e Collaborative System requires more than just these basic features and these are yet to be developed. Also, our current system must be improved, adding the features described in Section 5.

The developments of the Grid area will have to be closely monitored, as the switch from OGSI to WSRF will very likely make our current implementation

incompatible with the new toolkit, though we believe the effort required to change an OGSI-based system to WSRF will be small.

References

1. Afsarmanesh, H., et al. VLAM-G: A Grid-Based Virtual Laboratory. Scientific Programming: Special Issue on Grid Computing, volume 10, number 2, p.173-181. 2002
2. Baker, K., Greenberg, S. and Gutwin, C. Heuristic Evaluation of Groupware Based on the Mechanics of Collaboration. Proceedings of the 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01). (May 11-13, Toronto, Canada). 2001
3. Java Shared Data Toolkit User Guide http://java.sun.com/products/java-media/jsdt/
4. Bradner, E. and Mark, G. Why Distance Matters: Effects on Cooperation, Persuasion and Deception. In proceedings of Computer Supported Cooperative Work (CSCW 2002). November 16-20, New Orleans, Louisiana. 2002
5. The Globus Aliance http://www.globus.org
6. Gutwin, C. and Greenberg, S. The Mechanics of Collaboration: Developing Low Cost Usability Evaluation Methods for Shared Workspaces. Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, p.98-103, June 04-16, 2000
7. Grudin, J. Why groupware applications fail: Problems in design and evaluation. Office: Technology and People, 4 (3). 245-264. 1989
8. Kaletas, E. C. Scientific Information Management in Collaborative Experimentation Environments. PhD Thesis. University of Amsterdam, Faculty of Science. 2004
9. Korkhov, V., Belloum, A., Hertzberger, L. O. VL-E: Approaches to design a Grid-based Virtual Laboratory. 5th Austrian-Hungarian workshop on distributed and parallel system, September 19-22, 2004
10. de Ridder, A., Belloum, A. S. Z. Collaboration in the Virtual Laboratory for e-Science. 2004
11. Usability First http://www.usabilityfirst.com/groupware/index.txl (Groupware)
12. WTCT NV Virtual Lab e-Science: Towards a new Science paradigm. 2003
13. Stevens, M. http://www.developer.com/design/article.php/1010451 (Service-Oriented Architecture Introduction)

A Fully Decentralized Approach to Grid Service Discovery Using Self-organized Overlay Networks

Qi Xia[1,2], Weinong Wang[2], and Ruijun Yang[1,2]

[1] Department of Computer Science and Engineering,
Shanghai Jiaotong University, Shanghai, China
xiaqi@cs.sjtu.edu.cn
[2] Network Center,
Shanghai Jiaotong University, Shanghai, China
{wnwang,rjyang}@sjtu.edu.cn

Abstract. Self-organized overlay is widely used in Peer-to-peer (P2P) networks for its scalability, adaptability, and fault-tolerance in large and dynamic environments. In this paper, we propose a fully decentralized approach, DGSD(Decentralized Grid Service Discovery) for Grid service discovery. DGSD makes use of the underlying P2P overlay network protocols to self-organize nodes and services in the Grid. In DGSD, Grid service is represented by (attribute, value) pairs, and the Grid community, named virtual organization (VO), is self-organized into 2-dimensional overlay networks. One for attributes, and the other for values. Service discovery request in a VO is firstly directed to the attribute overlay network to find the servers with the searched attributes, then to the value overlay until the servers which have the same (attribute, value) pairs are reached. The architecture of DGSD supports interactions between VO's. Our approach is highly efficient, scalable and quickly responsive.

1 Introduction

Recent years have seen the rapid development of Grid technologies [1]. Many Grid projects are coming into use, which tend to increase the population in the Grid, and carry more heterogeneity and dynamic behaviors.

Open Grid Service Architecture (OGSA) [2] defines an open and service-oriented architecture to allow the interaction of Grid services. Therefore, the service discovery in the Grid is among the principle challenges. Globus implement the service publish/discovery mechanism based on MDS-2 [2, 3] which uses centralized register server. Although MDS-2 solved the scalability problem using hierarchical architecture, it is still vulnerable to single point of failure. Moreover, adaptation to the dynamic feature of servers is another challenge for MDS-2. Therefore, the natural way is to use decentralized approach for service discovery.

Recently, P2P communities have developed a number of fully decentralized protocols, such as CAN [4], Chord [5], and Pastry [6], for routing and

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 164–172, 2005.

lookup in P2P networks. The core idea behind the protocols is to build self-organized overlay networks when nodes join. In such overlay networks, each node or the published item will be given a unique ID using Distributed Hash Table (DHT). Motivated by these works, we propose a new decentralized approach, DGSD (Decentralized Grid Service Discovery), for Grid service discovery.

The original DHT based overlay networks only support queries for items which are expressed by one keyword. In the context of Grid service, we represent the Grid service by a number of (attribute, value) pairs. Therefore, straightforward DHT indexing for the services is not applicable. However, if the (attribute, value) pairs are placed into two search space, one for attributes, and the other for values. In each space, we implement the DHT protocol into every space to build overlay networks, then the problem can be solved. In addition, since in the Grid infrastructure, service provider and consumer are organized into Virtual Organizations (VO). We comply with such rule to develop DGSD.

The remainder of the paper is as follows. Section 2 surveys the related work. Section 3 propose the overview of the architecture of DGSD. Section 4 describes the detailed process of service discovery through DGSD. Perform analysis is done in section 5, and finally, section 6 concludes the paper.

2 Related Work

There are many projects undergoing address the issue of service discovery in the Grid. Globus [2] proposes MDS-2 for service discovery. MDS-2 provides a directory service based on Lightweight Directory Access Protocol (LDAP), where service is defined by a set of entities expressed by attribute-value pairs. The Grid services are integrated in Grid Information Index Servers (GIIS) [2], which includes GRRP (Grid Resources Registration Protocol) to let all services to be registered, and GRIP (Grid Resource Information Protocol) to allow access to the registered services.

The work related to MDS-2 is Web service [7]. Web service uses WSDL (Web Service Description Language) [8] to describe services, and UDDI (Universal Description, Discovery and Integration) [9] is applied for service registration and discovery.

The above approaches use centralized registration servers for all services. In such infrastructures, querying for services is very efficient and complex query can be supported well. However, for a very large and dynamic environment, they face the problems of scalability and single point of failure.

A natural and easily implemented improvement is to introduce hierarchical overlay network to distribute the registration and query into network members, as is done by [10] and [11]. They organized the nodes in the network in a hierarchical way. In [10], nodes join and search in the hierarchy by flooding. And in [11], nodes firstly self-organize themselves into a tree-like hierarchy, and the query message is firstly directed to the top level node, then flood to all the underlay level. The main drawback of these approach is the network traffic incurred,

and the single point of failure of the node at the top of the hierarchy will make the query to fail.

The first work addressing fully decentralized service discovery in the Grid was done by Iamnitchi et al [12]. However, the algorithm they used is forward-and-flooding, which is not efficient. Recent study on routing in P2P networks give indication for Grid service discovery. Using DHT based protocols such as CAN [4], Chord [5], or Pastry [6], all the nodes self-organize themselves into overlay networks. The overlay network is fault-tolerant, and searching in such network is scalable and efficient. However, the original DHT protocols only support (key, value) mapping, and one node or item only has one key. In case of Grid service, which is expressed by multiple (attribute, value) pairs, it's hard to implement the DHT in that way.

To expand the original DHT mechanism, some systems were developed to support service discovery in different environment. INS/Twine [13], which is used to discover web service in a peer-to-peer way, split service description into multiple strands - the keywords. All strands are indexed into a common overlay network using Chord protocol, thus the service can be discovered using multiple keywords. Lanes [14] proposes an approach of service discovery in mobile Ad Hoc network based on CAN protocol. And [15] implemented a range query mechanism for Grid information. These works are close to our approach in that overlay network is used and adaptation is applied to the underlying P2P protocol to support indexing complex service descriptions, and are efficient and scalable. However, the approach of ours is more intuitive, and we explicitly address the use of VO.

3 The Architecture for Grid Service Discovery Using Overlay Based on VO

In the Grid, all service providers and consumers are organized into VO's. Each VO contains miscellaneous members, and resources are described by services, which can be represented by multiple (attribute, value) pairs. For example, the following description of service:

```
<res>computer
  <OpSys>UNIX</OpSys>
  <Mem>512</Mem>
</res>
```

refers to a computer which runs the UNIX operating system, and provides 512 MByte memory for computing. It can be decomposed as three attribute-value pairs: (res, computer), (OpSys, UNIX) and (Mem, 512). When one server wants to join and publish its service in a VO. It firstly joins an overlay network in the attribute space using DHT based protocol, and publish all the attributes in the space respectively. Meanwhile, upon publishing each attribute, the server also asks the current server in charge of the attribute to publish the value corresponding to the attribute, i.e., there is an another overlay network for values. Fig. 1

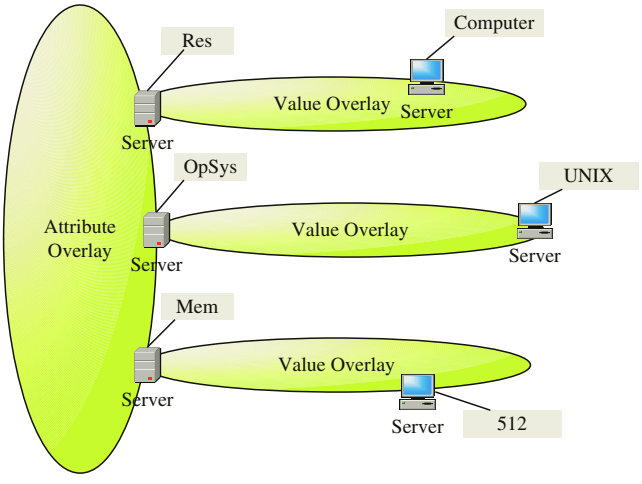


Fig. 1. Overview of the service overlay networks. Each service provider(denoted by Server) in the network joins two overlay network, one is attribute overlay, the other is value overlay. In the figure, an example of computing service as described before is shown. The attribute server plays a role as entry point of the value overlay with the corresponding attribute

shows the overview of the overlay networks. When a consumer (it may also be a server) issues a service query message, it firstly gets the (attribute, value) pairs from the service description, and contacts one server in the VO. Then for every attribute, it locates the server in charge of the attribute in the attribute overlay, then asks it to find the server in charge of the corresponding value. Searching for every (attribute, value) pair is performed in a parallel way, and there may be a number of matching servers for every pair returned to the consumer. Finally, the consumer selects the server matching all (attribute, value) pairs.

All the servers join the overlays only within their own VO. Notice that a server may join not only one VO, then it may join multiple overlays lie in different VO's. Service query will be executed in one's own VO firstly. If there is no matching service, the query message will be routed to the gateway server of the VO, and redirected to other VO's which have connection with it. Fig. 2 shows the overall architecture.

4 Service Discovery Using DGSD

In the following, we give formal definitions used in DGSD and the detailed process of the service discovery.

Definition 1. *Attribute overlay network (denoted by AON) is a self-organized overlay network constructed by servers in a VO. And* AON_V *denotes the overlay in VO named* V*.*

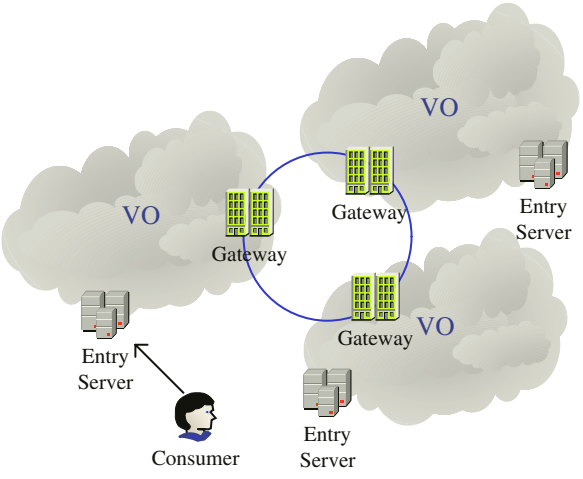


Fig. 2. The overall architecture for the interaction among VO's. The gateway server may be some well-known server, and entry server is the entrance point for consumers which have not joined the VO

Definition 2. *Value overlay network (denoted by VON) is a self-organized overlay network constructed by servers in a VO. And* VON_V^{α} *denotes the overlay corresponding to attribute* α *in VO named* V*.*

Definition 3. *Gateway servers (denoted by* GS_V*) in VO named* V *are the connection points to other VO's.*

Definition 4. *Entry servers (denoted by* ES_V*) in VO named* V *are the entrance points for consumers not in* V*.*

Table 1 describes the application programming interface (API) provided by DGSD.

Table 1. The API in DGSD

Function	Description
nodeInit(*credentials*,V)	To initialize status of a node in V, and return nodeId
generate(*serv*)	According to service description of *serv*, generate (attribute, value) pairs.
route(*msg*,*key*,*net*)	In network *net*, find the node with nodeId numerically closest to the key using the underlying DHT protocol, and return the node.
join(*xON*,*key*,*type*)	A server with *key* joins *xON* (either AON or VON) using the underlying DHT protocol by contacting an arbitrary node in *xON*; Or a node wants to join in *xON*. *type* (either node_joining or key_publishing) directs the operation executed by nodes along the routing path.
reroute(V,*msg*)	reroute a message *serv* to V by the Gateway Server.

4.1 Service Discovery Within VO

In section 3, we have shortly described the process of service discovery within a specific Grid VO. Here we use the predefined API to give the detailed description.

Firstly, each service provider should publish its service in a VO.

```
//publish service in VO named V
server.publish(V,serv)
{
  generate(serv)→{α_i, v_i};
  for each attribute α_i
    join(AON_V, α_i,key_publishing);
    join(VON_V^{α_i}, v_i,key_publishing);
}

//join operation is for publishing one key or node joining
server.join(xON,key,type)
{
  (key,server,type)→msg;
  if type=node_joining
    nodeInit(credentials,V)→nodeId;
    key:=nodeId;
    route(msg,key,xON)→Node;
    constructing routing table and joining xON, using the underlying DHT protocol
  else if type=key_publishing
    route(msg,key,xON)→Node;
    Storing key and server information in Node;
}
```

Then service discovery can be performed for all published services.

```
//discover services in VO named V, assuming the consumer is also in the VO
consumer.discovery(V,serv)
{
  generate(serv)→{α_i, v_i} (i = 1, 2, ..., k);
  for each attribute α_i
    (α_i, v_i, discovery)→msg;
    route(msg,α_i, AON_V)→ N_i;
    N_i.route(msg,v_i, VON_V^{α_i})→ N_i;
    (servers corresponding to (α_i, v_i) stored in N_i) → S_i;
  return S_1 ∩ S_2 ∩ ... ∩ S_k;
}
```

4.2 Service Discovery Between VO's

We consider a more complicated case. when a consumer not in a VO wants to discover one Grid service, it will contact the entry server of the VO to perform

service discovery. If there is no service found in the VO, the discovery message will be directed to other VO's.

```
//consumer is not in the VO
consumer.discovery(V,serv)
{
  contact ES_V, send query message with serv;
  ES_V.discovery(V,serv)→ S;
  if S = φ
    ES_V send query message to the gateway GS_V;
    (serv,discovery)→msg
    GS_V.reroute(V',msg)→ GS_V';
    GS_V'.discovery(V',serv)→ S';
    S' → GS_V → ES_V;
    return S';
  else
    return S;
}
```

Notice that the gateway server may have to contact not only one other VO to perform the discovery.

5 Performance Analysis

Firstly, we evaluate the storage load(L) on all nodes in the VO. The average load on a node is:

$$L_{avg} = \frac{2VS}{N},$$

where S is the number of services provided, V is the average number of (attribute,value) pairs each service, which is typically $3 \sim 5$, and N is the number of service providers in the VO. The factor 2 in the equation appears because for each (attribute,value) pair, we have to publish both the attribute and value in AON and VON separately. There may be some popular services, such as file, which will cause very high load on the node in charge of file attribute. To resolve this problem, we use the similar mechanism as [13], i.e., we set a load threshold for a node. If the load on the node overloaded over the threshold, it may select another node to store the items, while maintaining a link to it.

We use CAN [4] as the underlying overlay network protocol due to its potential support for range query [15, 16]. In a overlay network, each node maintains a routing table containing d other nodes, and the routing latency to any node is $O(dN^{1/d})$. Therefore, from the algorithm defined in section 4, the discovery latency in a VO is $O(2dN^{1/d})$, because the discovery is performed in parallel for all (attribute,value) pairs of a service, and for each pair there must be two sequentially discovery, one for the attribute, the other for the value. The average routing table size for a node is $2dV$, where V is the average (attribute, value)

pairs. We do not give the detailed simulation results here since it's the work of the underlying overlay protocol.

Replication mechanism should be implemented to ensure successful discovery. In our architecture, nodes in AON serves as entrance for VON. If some node in AON fails to be responsive, the discovery for the corresponding (attribute, value) pair will fail. Therefore, for each node corresponding to some attribute in AON, the successor or predecessor information in the corresponding VON must be replicated into some other nodes, so that when it fails, the discovery message can be routed to those nodes then enter the VON to continue value search. Similar replication mechanism should be also applied in VON.

In this paper, we only give an example of exact service discovery. The original CAN does not support partial or range query. However, using Hilbert space filling curve (SFC) [17] instead of hash operation to numbering nodeId, attributes and values. The underlying P2P protocol can support partial query similiar to [18], which used SFC to map multi-dimensional data items into 1-dimensional continuous numbers and still maintain the locality of the original data items. And range query [15] is also supported well.

Relying on the underlying overlay network protocol, DGSD is highly efficient, scalable, and fault-tolerant, and can be adaptable to the dynamic environment to be quickly responsive to the changes of node status.

6 Conclusion

Service discovery in the Grid should be scalable, efficient, fault-tolerant, and quick responsive to the dynamic changes of the nodes. DGSD provides such an approach. The key idea behind DGSD is that service can be decomposed as (attribute,value) pairs, and will be published into a 2-dimensional overlay networks for both attribute and value. To build the 2-dimensional overlay networks, we make use of the state-of-the-art P2P overlay protocols, which is proven suitable in large and dynamic environments. Fully decentralized service discovery in the Grid is highly deserved especially when the population grows very large and the behavior of the population is very dynamic. DGSD can supplement MDS-2 to improve the performance or be deployed independently.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal on Supercomputing Applications **15** (2001)
2. Foster, I., et al, C.K.: The physiology of the grid: An open grid services architecture for distributed systems integration (2003)
3. Foster, I., Kasselman, K.: The globus project: A status report. In: IPPS/SPDP'98 Heterogeneous Computing Workshop. (1998)
4. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. ACM SIGCOMM, San Diego, CA (2001)

5. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. ACM SIGCOMM 2001. (2001) 149–160
6. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). (2001) 329–350
7. Web services for business process design, http://www.gotdotnet.com/team/xml-wsspecs/xlang-c/default.htm (2004)
8. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (WSDL) 1.1. w3c, http://www.w3.org/tr/wsdl (2002)
9. UDDI: Universal description, discovery and integration, http://www.uddi.org/ (2004)
10. Crespo, A., Garcia-Molina, H.: Semantic overlay networks for p2p systems. Technical report, Computer Science Department, Stanford University (2002)
11. Huang, L., Wu, Z., Pan, Y.: A scalable and effective architecture for grid services discovery. In: Proceedings of SemPGRID'03. (2003)
12. Iamnitchi, A., Foster, I.: On fully decentralized resource discovery in grid environments. In: International Workshop on Grid Computing, Denver, Colorado, IEEE (2001)
13. Balazinska, M., Balakrishnan, H., Karger, D.: INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In: Proceedings of Pervasive. (2002)
14. Klein, M., Konig-Ries, B., Obreiter, P.: Lanes - a lightweight overlay for service discovery in mobile ad hoc networks. Technical Report 2003-6, University of Karlsruhe (2003)
15. Andrzejak, A., Xu, Z.: Scalable, efficient range queries for grid information services. In: Proceedings of the Second IEEE International Conference on Peer-to-Peer Computing (P2P2002). (2002)
16. Tang, C., Xu, Z., Mahalingam, M.: Peersearch: Efficient information retrieval in peer-peer networks. Technical Report HPL-2002-198, Hewlett-Packard Labs (2002)
17. Asano, T., Ranjan, D., Roos, T., Welzl, E., Widmaier, P.: Space filling curves and their use in geometric data structures. Theoretical Computer Science **181** (1997) 3–15
18. Schmidt, C., Parashar, M.: A peer-to-peer approach to web service discovery. World Wide Web **7** (2004) 211–229

Dynamic Parallelization of Grid–Enabled Web Services*

Manfred Wurz and Heiko Schuldt

University for Health Sciences, Medical Informatics and Technology (UMIT),
Information and Software Engineering Group,
Eduard-Wallnöfer-Zentrum 1, A–6060 Hall in Tyrol, Austria
{manfred.wurz, heiko.schuldt}@umit.at

Abstract. In a grid environment, it is of primary concern to make efficient use of the resources that are available at run-time. If new computational resources become available, then requests shall also be sent to these newly added resources in order to balance the overall load in the system. However, scheduling of requests in a service grid considers each single service invocation in isolation and determines the most appropriate provider, according to some heuristics. Even when several providers offer the same service, only one of them is chosen. In this paper, we provide a novel approach to the parallelization of individual service requests. This approach makes dynamic use of a set of service providers available at the time the request is being issued. A *dynamic* service uses meta information on the currently available service providers and their capabilities and splits the original request up into a set of simpler requests of the same service types, submits these requests in parallel to as many service providers as possible, and finally integrates the individual results to the result of the original service request.

1 Introduction

Grid computing aims to establish highly flexible and robust environments to utilize distributed resources in an efficient way. This can be, for example, computational resources, storage capacity, or various external sensors. An essential feature of grid environments is to make use of the resources that are available at run-time. While data grids focus on the exploitation of storage resources, *service grids* mainly consider computational resources for scheduling. In particular, if new computational resources become available, then requests will also be sent to these newly added resources in order to balance the overall load in the system.

The advanced resource management of service grid infrastructures seamlessly considers web service standards and protocols for making application logic accessible. Web services can be invoked by common web protocols (e.g., SOAP over

* This work has been partially funded by the EU in the 6^{th} Framework Programme under the network of excellence DELOS (contract no. G038-507618) and by the Austrian Industrial Research Promotion Fund (FFF) under the project HGI

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 173–183, 2005.

HTTP) and are described in a platform-independent way by XML and WSDL. All this has led to the recent proliferation of web service technology which has also gained significant importance in the grid computing community.

However, scheduling of (web) service requests in a grid considers each single service invocation in isolation and determines the most appropriate (web) service provider, according to some heuristics that, for example, take into account the current load of all providers of this particular service in the overall system. Although several providers offer the same service or at least semantically equivalent services, only one of them is chosen at run-time and the request is submitted to this provider for processing. This distribution is even independent of the complexity of the service request in question.

In this paper, we provide a novel approach to the parallelization of individual web service requests by making dynamic use of a set of providers of services of the same type which are available at the time the request is being issued. This work specifically focuses on powerful new services using composition, self-adaptability, and parallel service execution. The contribution of this paper is to introduce an architecture of a service seeming to be an ordinary, callable service to the outside world, which is able to adopt its behavior based on some quality of service criteria attached, and the resources available on a grid. In short, this *dynamic* service uses meta information on the currently available service providers and their capabilities (taken from a service repository) and splits the original request up into a set of simpler requests (in terms of the data that has to be processed) of the same service types, submits these requests in parallel to as many service providers as possible, and finally integrates the individual results from these service providers to the result of the original service request.

The following scenario illustrates in which way dynamic services can be used to solve real world problems, and how easily they can be integrated in existing infrastructures. The applicability to large scale digital library systems within a healthcare environment has been presented in [1].

Clustering: *Data mining in high dimensional feature spaces is a commonly used approach to gain new knowledge in medical informatics and bioinformatics. In the field of functional metabolomics, it is, for example, used to support the identification of disease state metabolites without any prior knowledge and permits the construction of classification rules with high diagnostic prediction [2]. In this type of applications, clustering is important to understand the natural structure or grouping in a data set. Clustering, in particular, aims at partitioning the data objects into distinct groups, maximizing the similarity within that group, while minimizing the similarity between groups. Finding clusters in high dimensional feature spaces is a computationally intensive task (more details can be found in [3]). SURFING (SUbspaces Relevant For clusterING) [4], a sample clustering algorithm, computes a distinct quality measure per subspace and then ranks them according to the interestingness of the hierarchical clustering structure they exhibit. In worst case, this algorithm has to consider all* 2^d *subspaces (where d is the dimensionality of the feature space). Using SURFING, the number of relevant subspaces within the whole data set can be significantly reduced:*

for most complex data sets, only 5% *of all* 2^d *subspaces have to be examined [4]. This benefit is achieved by calculating a quality measure based on the k-nearest neighbor distances (k-nn-distances), which however has to be done in* $O(n^2)$ *(n being the number of feature vectors to examine), leading to an overall complexity of* $O(2^d \cdot n^2)$*. Since these* $O(n^2)$ *calculations are independent, they are good candidates for (dynamic) parallelization. Assuming the availability of m instances of a service to calculate k-nn-distances, the total effort for the subspace clustering can be reduced to* $\frac{0{,}05 \cdot O(2^d \cdot n^2)}{m}$ *(ignoring the effort to distribute the data set examined). When knn-distance calculation is available as service by different providers, it is highly beneficial to dynamically incorporate as many instances as possible for improving the complexity of a clustering algorithm.*

In this scenario, we assume that the implementation of the actual service is already completed, and focus on the provision of dynamically adapting services that parallelize execution. These services will provide added value to existing applications which can profit from parallel execution without touching the conscientiously tested business logic itself.

The remainder of this work is organized as follows. Section 2 introduces the concepts and components involved in dynamic call distribution. In Section 3, a concrete implementation is presented and Section 4 provides first experimental results. Related work is discussed in Section 5. Section 6 concludes.

2 A Dynamically Adapting Service for Parallel Execution

The combination of individual, generally usable services to solve complex and specialized problems is of great importance in most applications. These efforts mainly concentrate on abstract workflow definitions which can than be deployed to the grid, and be bound to specific resources at the latest possible time. A particular workflow step, a task that has to be processed, is then always mapped to the service or resource, which best fits the requirements. If more than one service is able to fulfill the requirements, one of them is chosen, following the one or the other load balancing algorithm.

In our proposed architecture, we put the emphasis on improving the usability of a single service, as well as on enabling faster and less error prone development for grid environments. This approach is based on the observation, that following the current proliferation of service oriented architectures, the number of services and service providers in a grid will significantly increase. Especially services which are provider independent and are not bound to special resources can be distributed fast and widely in a grid environment or be deployed numerously on demand.

The task of partitioning request parameters and reintegrating results afterwards is highly application specific and, from our perspective, can not be solved in a generic way. Although we see the potential to identify classes of applications according to the mechanism they partition and reintegrate requests which allows to have pre-built splitter and merger services, an expert in the problem domain

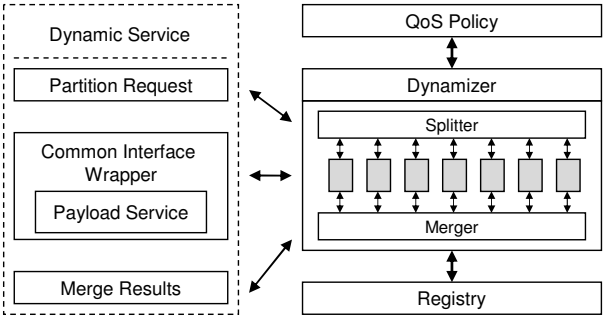


Fig. 1. Overall Architecture

will be necessary to tailor them for the specific need or perform some additional, application domain specific work.

In our approach, splitting requests and merging results, as well as looking up available services in a registry, is transparent to the user. The service that is i.) enhanced with knowledge of how to partition the requested task into subtasks, and ii.) how the partial results can be re-integrated, can still be accessed as before. We call a service enhanced that way a *dynamic service*.

As shown in figure 1, the following logical units can be identified for dynamic services.

The box in the center of the left side, labeled 'Payload Service', represents the actual service. It is responsible for providing application semantics, e.g., a complex computation or a database lookup. This is usually a piece of business logic that has existed beforehand, which is now supposed to be opened to the grid and enabled for parallel execution. To achieve this goal, it is surrounded by another box, labeled 'Common Interface Wrapper', which encapsulates the 'Payload Service' and enhances it with a common interface.

On top, 'Partition Request' encapsulates knowledge on how incoming parameters for the 'Payload Service' have to be partitioned, so that the original request can be decomposed into numerous new 'sub'- requests. Each of these 'sub'- requests can than be distributed on the grid, and be processed by other instances of the originally targeted service. The box at the bottom ('Merge Results') integrates (partial) results returned from the grid to provide the original service requester with a consolidated result. It can therefore be seen as the reverse operation to the 'Partition Request' service. The combination of these elements is referred to as 'Dynamic Service'.

To find the instances of the originally targeted service (e.g., services where the functional description equals the one of the 'Payload Service'), a registry is used (depicted in the lower right corner of figure 1) . This registry provides information on which services are available, how they can be accessed, and what their properties are (in terms of CPU load, connection bandwidth, access restrictions, etc).

The 'Dynamizer', depicted on the right hand side, makes use of the services mentioned above. It glues together the previously described services by making the parallel calls and coordinating incoming results. It has also to provide appropriate failure handling. It is, in contrast to 'Partition Request' and 'Merge Results', application independent and generally usable. The 'Dynamizer' can interact with all services that adhere to a common interface, as ensured by the 'Common Interface Wrapper'. It can be integrated in environments able to call and orchestrate services, or it can be packaged and deployed together with specific services.

To make the best possible use of the 'Dynamizer', the user can send a description of the desired service quality along with the mandatory parameters needed to process the request. In this QoS (Quality of Service) policy, the user can, for example, describe whether the request should be optimized in terms of speed (select high performance nodes, and partition the input parameters accordingly), in terms of bandwidth (try to keep network usage low) or if it should aim for best accuracy (important for iterative approaches or database queries, where there is an option to use different data sources). Since these specifications can be contradictory, adding preferences to rank the users requirements is of importance. To better illustrate the mechanisms within the 'Dynamizer' regarding the user specified QoS policy, we consider the following example: A scientist wants to use the SURFING algorithm as described in section 1 to examine a set of metabolomic data. In the QoS policy file, he specifies that network usage should be kept low (because his department has to pay for each megabyte sent through the wire), and as a second preference to have his call optimized in terms of speed. The 'Dynamizer' has three powerful computational services at hand, which would be able to deliver the result within approximately two hours at the cost of three times transferring the data set, or, alternatively, 400 less powerful services, which would be able to deliver within 20 minutes but at the obvious cost of much higher network usage. The algorithms on how to reconcile the users specifications, the details of the QoS description language and how to integrate this best with our existing implementation is currently under investigation.

3 Implementing Virtual Dynamic Web Services

3.1 Dynamizer

The most vital part, the hub, where all the main control flow, the intelligence and the failure handling, is located, is within the so called 'Dynamizer' (shown in figure 1). It is responsible for issuing the calls in parallel, in our implementation each one in a separate thread, collecting the results, and combining them to match the original request.

If a request to a 'dynamic' service (consisting of the actual service and enhanced with a 'Common Interface Wrapper', the 'Partition Request' and 'Merge Results' services) is issued, it is redirected to the 'Dynamizer'. The registry is queried for a list of available instances that are as well able to process the re-

quest. In case there are any, the original request can be decomposed using the 'Partition Request' service attached to the *'dynamic'* service. In our case, this is implemented as an additional operation within the originally called web service, named *splitParameters*. This operation takes a list of input parameters, the ones specified by the issuer of the original call, along with a parameter indicating how many partitions should be created. The decision on how many partitions should be created is made by the 'Dynamizer', based on information provided by the registry. If the *splitParameter* operation can not produce as many partitions as asked for by the 'Dynamizer', it is empowered to adapt that parameter in favor of producing an error. Along with the partitions of parameters, it creates a re-assembly plan, which is later on used to reconstruct the overall result.

Having a number of available services, as well as the partitioned parameter set at hand, the 'Dynamizer' issues parallel calls to those services, each with one of the partitions as an input. If the number of available services does not match the number of partitions, not all available services are used, or some are used more then once, respectively. Alternatively, a surplus of services can be used to backup others, in case the nodes hosting the services fail or are disconnected (or are known to be less reliable than others).

Finally, the (partial) results returned are integrated using the service 'Merge Results'. It is, like the *splitParameters* operation, included in the *'dynamic'* service. In our implementation, it was realized as an operation named *mergePartitialResult*. It accepts as an input a reference to the overall result, the (partial) result returned by one of the service instances called and the re-assembly plan produced during partitioning. According to this re-assembly plan, the partial result is inserted into the final result. When all partial results are returned, the 'Dynamizer' forwards the overall result to the original issuer of the call. Figure 2 shows a sequence diagram to illustrate the call sequence among the described services.

3.2 Dynamic Service

In contrast to the 'Dynamizer' who plays a managing and coordinative role, the *'dynamic'* service exposes a piece of business logic to the grid, enhanced with the possibility to dynamically execute incoming calls in parallel. It achieves that by adding two additional operations, *splitParameters* and *mergePartitialResult*, and interacting with the 'Dynamizer' (as described above). To present all possible *'dynamic'* services to the 'Dynamizer' with one common interface, the actual service is wrapped within an operation we termed *doOperation* in our implementation. In the architectural view depicted in Figure 1, it is labeled with 'Common Interface Wrapper'. It accepts partitions of parameters as an input, can do some type conversions if necessary and maps the partition to the parameters of the actual service.

Revisiting the scenario from chapter 1, the splitting can easily be achieved by assigning each available knn-distance service a fraction of distances to calculate. Let there be m distance calculation services available and a set of n feature vectors, then each calculation service will have to return the knn-distances for

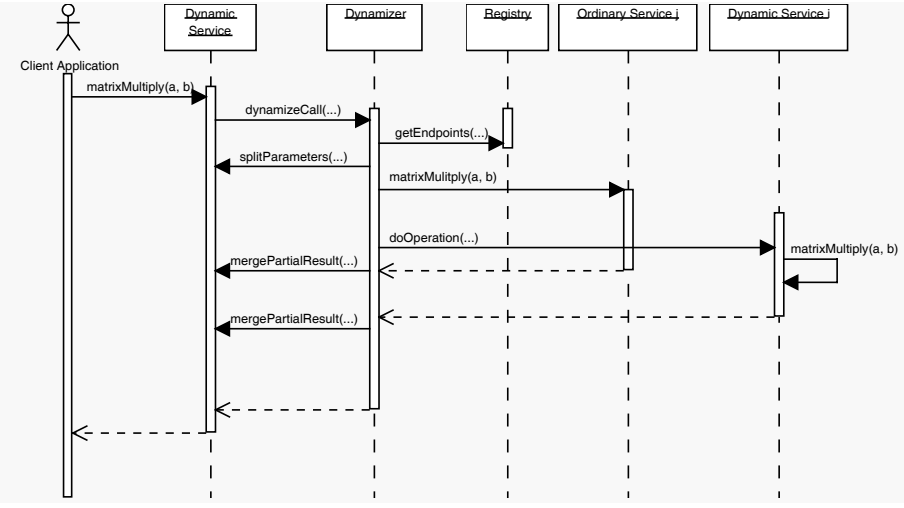


Fig. 2. Sequence Diagram of Service Invokation

n/m feature vectors. This ordered result vectors can then be merged into the overall result vector of n knn-distances.

Depending on the actual service(s) in terms of number of available instances, load, speed of instances available etc., and the QoS policy specified by the user, different kinds of decompositions might be fruitful. If, for example, bandwidth usage has to be kept small or the network used is slow, partitioning might occur differently at the cost of less computational speed. The intelligence about which way of decomposition is best is currently left to the implementation of the 'Split Request' service – but better support by the infrastructure is subject of ongoing research.

3.3 Registry

As registry, any catalog service that is capable of storing information about services available in the grid together with some metadata about their potential behavior can be used. We have implemented a simple registry to store data about the services available in our test bed, but others as GT3's IndexService or a UDDI server could of course be used as well.

4 Experiments and Results

We have implemented a simple matrix multiplication service which can be used for dynamic parallelization. We ran it on varying numbers of personal computers serving as our grid test bed. We have exploited ordinary desktop computers with either Linux or Windows as operating system, and Tomcat with Axis as basis

Table 1. Processing time for multiplication of NxN matrices on up to 4 PC's

	4 Nodes	3 Nodes	2 Nodes	Non Parallel
$N = 60$	1990 ms	2200 ms	2650 ms	3940 ms
$N = 120$	5030 ms	5410 ms	6440 ms	7500 ms
$N = 180$	9580 ms	9720 ms	11390 ms	11250 ms
av. speedup	1.5	1.4	1.2	1.0

for call distribution. The 'Payload Service', the matrix multiplication, was by itself not aware of participation in a distributed infrastructure. It was just an ordinary method implemented in Java, able to multiply two matrices passed in as arguments and return the resulting product.

To enable this method for use through the previously described 'Dynamizer', the two additional services 'Partition Request' and 'Merge Result' have been implemented. To comply with the strategy that the payload services itself should not be changed, all parameter partitions generated by the 'Partition Request' service have to be regular inputs for a matrix multiplication. Therefore, the partitioner cuts the first matrix along its rows into equally sized parts. Assume a matrix having m rows, n columns and k computers offering the multiplication service at the time of partitioning the request, this results in k matrices each having m/k rows and n columns. The second matrix is partitioned analogously. Along with this sub matrices, a description is generated for each partition which stores the information of where to integrate the partial result. The 'Merge Result' service copies the partial results received, according to the re-integration description, into the result matrix.

Table 4 shows the results of multiplying $N \times N$ matrices, consisting of randomly generated integer values out of the set $0, 1$, locally or distributed on up to four PC's using dynamic partitioning and distribution of calculation requests as described. To better shape out the usage of this distribution pattern for computational intensive tasks, the multiplication was interrupted for 50 ms per resulting row.

It can be seen from the results in Table 4 that the speedup by adding nodes to the system has not been tremendous. But we would like to stress that the actual speedup of execution was not the primary goal in this work. Rather, the dynamics of parallel execution, partitioning, and merging have been our main focus. To gain better performance, optimizations can be applied in the algorithm to partition requests. Similarly, the data that has to be transferred can be additionally compressed. The up to four PC's used have been added or removed from the grid without changing a single line of code in our services. Nevertheless, the components adapted themselves automatically to the new environment.

5 Related Work

There are numerous possibilities to decompose an application into smaller parts that can then be executed in parallel. One of the most important and widely

known decomposition pattern is to use a central master coordinating control and data flow, and several slaves executing sub tasks. Although there are other possibilities like divide and conquer or branch and bound, the master/slave paradigm is especially suitable for grid environments [5] and therefore widely used. The master worker tool [6] allows to integrate applications in the grid by implementing a small number of user-defined functions similar to the approach described in this paper, but has a strong focus on problems from the field of numerical optimization [7]. While the master worker tool is tightly integrated in a Globus Toolkit 2 environment, our approach focuses on evolving into a more generally usable framework and is independent of the underlying grid infrastructure.

Similarly, the AppLeS Master-Worker Application Template (AMWAT) [8] offers a mature library to ease the creation of applications which are able to solve a problem by breaking it into subproblems and merging the subproblems results into an overall solution. AppLes emphasizes scheduling and fault tolerance issues. In contrast to the explicit exploitation of AppLeS agents and necessary adaptation of existing applications, we aim to do the parallelization transparently and by wrapping existing code instead of interweaving it. Work with other task parallel models can be found in [9, 10] using divide and conquer mechanisms and [11] for an example of branch and bound type of decomposition.

Using Java [12] to build environments for parallel and distributed environments and research about performance differences to other technologies was, among others, conducted in [13][14]. In [13], the Java Language has been enriched with a set of constructs like remote object creation, remote class loading, asynchronous remote method invocation, and object migration focusing on 'java-only' environments, the evolution of web services enabled us to easily integrate all environments being able to invoke web services. In [14], a good overview on various programming models for parallel java applications can be found.

In addition to the possible registries mentioned in the previous chapters, Miles et. al. describe a service registry which allows to store semantically enhanced services [15]. It extends the standard UDDI interface to provide semantic capabilities by attaching metadata to entities within a service description.

6 Conclusion and Outlook

In this paper, we have presented a novel approach to the dynamic parallelization and automatic adaptation of (web) service calls. Invocations of a service that is extended to be dynamic are split up at run-time into a set of sub-requests which are sent in parallel to different service providers. After execution, the results of the sub-requests are integrated in order to determine the result of the original service call. This allows for the fast development of distributed, standards-based parallel applications. In many cases, it enables parallel execution of readily developed and deployed business logic without changes to existing code.

In further work we plan to implement the assignment of QoS policies to services requests. Currently, we are working on the specification of a language to formulate these policies and on mechanisms that allow to apply these policies,

even in the case of contradictory specifications. The current implementation presented in this paper is searching for identical instances of the same service to distribute a call. An important question in our further work will be to examine ways to extend the search also to semantically equivalent services. Finally, as a next step, we aim to integrate our prototype implementation into OSIRIS (Open Service Infrastructure for Reliable and Integrated process Support) [16], a peer-to-peer process execution engine. This will allow to parallelize not only single service invocations but to consider dynamic parallelization of services in the context of process and workflow execution.

References

1. Wurz, M., Brettlecker, G., Schuldt, H.: Data Stream Management and Digital Library Processes on Top of a Hyperdatabase and Grid Infrastructure. In: Pre-Proceedings of the 6^{th} Thematic Workshop of the EU Network of Excellence DELOS: Digital Library Architectures - Peer-to-Peer, Grid, and Service-Orientation (DLA 2004), Cagliari, Italy, Edizioni Progetto Padova (2004) 37–48
2. Baumgartner, C., Böhm, C., Baumgartner, D.: Modelling of Classification Rules on Metabolic Patterns Including Machine Learning and Expert Knowledge. Journal of Biomedical Informatics, In Press (2004)
3. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Academic Press (2001)
4. Baumgartner, C., Plant, C., Kailing, K., Kriegel, H.P., Kröger, P.: Subspace Selection for Clustering High-Dimensional Data. In: Proc. IEEE Int. Conf. on Data Mining (ICDM'04). (2004)
5. Foster, I., Kesselman, C., eds.: The Grid 2, Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (2004)
6. Linderoth, J., et al.: An Enabling Framework for Master-Worker Applications on the Computational Grid. In: 9th IEEE Int'l Symp. on High Performance Dist. Comp., Los Alamitos, CA, IEE Computer Society Press (2000) 43–50
7. Anstreicher, K., et al.: Solving Large Quadratic Assignment Problems on Computational Grids. In: Mathematical Programming 91(3). (2002) 563–588
8. Shao, G.: Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources. PhD thesis, University of California - San Diego (2001)
9. Foster, I.: Automatic Generation of Self-Scheduling Programs. In: IEEE Transactions on Parallel and Distributed Systems 2(1). (1991) 68–78
10. v. Nieuwpoort, R., et al.: Efficient Load Balancing for Wide-Area Divide-And-Conquer Applications. In: 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. (2001) 34–43
11. Iamnitchi, A., et al.: A Problem-specific Fault-tolerance Mechanism for Asynchronous Distributed Systems. In: Int'l Conference on Parallel Processing. (2000)
12. Microsystems, S.: Java Technology. `http://java.sun.com/` (2004)
13. Izatt, M., Chan, P., Brecht, T.: Ajents: Towards an Environment for Parallel, Distributed and Mobile Java Applications. Concurrency: Practice and Experience **12** (2000) 667–685
14. Bull, M., Telford, S.: Programming Models for Parallel Java Applications. Technical report, Edinburgh Parallel Computing Centre, Edinburgh (2000)

15. Miles, S., Papay, J., Payne, T., Decker, K., Mureau, L.: Towards a Protocol for the Attachment of Semantic Descriptions to Grid Services. In: The 2nd European Across Grids Conference, Nicosia, Cyprus, Springer LNCS (2004)
16. Schuler, C., Weber, R., Schuldt, H., Schek, H.J.: Scalable Peer-to-Peer Process Management - The OSIRIS Approach. In: Proceedings of the 2^{nd} International Conference on Web Services (ICWS'2004), San Diego, CA, USA, IEEE Computer Society (2004) 26–34

Automatic Composition and Selection of Semantic Web Services

Tor Arne Kvaløy[1,2], Erik Rongen[1], Alfredo Tirado-Ramos[2], and Peter Sloot[2]

[1] IBM Center for Advanced Studies,
David Ricardostraat 2-4, 1066 JS Amsterdam, The Netherlands
torarnek@pvv.org, erik@nl.ibm.com
[2] Faculty of Sciences, Section Computational Science, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{alfredo, sloot}@science.uva.nl

Abstract. Interactive applications like Problem Solving Environments require on demand access to Web Services, where the services are autonomously discovered, composed, selected and invocated based on a description of requested capabilities. Semantic Web Services aim at providing semantically interpretable capabilities through the use of shared ontologies. We demonstrate how Grid Services for an interactive biomedical application are annotated with a domain ontology, and propose algorithms for automated composition and selection of workflows, where workflows are created by semantically matching service capabilities, and where workflow selection is based on a trade-off between the types of semantic matches in the workflow and the number of services. The algorithms are demonstrated on semantically annotated Grid Services in the biomedical application.

1 Introduction

Web Services are self-describing, self-contained units of application logic that are either used as a single service or composed into a workflow that is executed sequentially as a whole. Creation of such workflows has traditionally been accomplished by manually locating suitable services via a registry like UDDI and then composing them into a workflow. With the increase of available services and the dynamic nature of the Web it is desirable to automate these processes by having software agents discover, compose, select and execute workflows autonomously [1].

XML Schema based standards for describing Web Services (WSDL and SOAP) guarantee only syntactical interoperability between services in a workflow, while what is needed for automating the mentioned processes is interoperability on a semantic level. The Semantic Web provides standards for representing and reasoning with computer interpretable information [2], and this has lead to the development of OWL-S, which is an attempt to standardize how Web Services are described semantically, thus creating Semantic Web Services [3].

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 184–192, 2005.

OWL-S provides a semantic layer on top of WSDL that maps operation parameters to semantics defined in ontologies. Grid Services are with OGSI [4] and WS-Resources [5] based on Web Services in that the interfaces are defined with WSDL. OWL-S does neither reflect nor interfere with the standardized interfaces for statefulness, so we maintain that OWL-S can be used to describe the application specific interfaces of Grid Services. Semantic Grid Services are therefore in our view Grid Services with semantically annotated capabilities, where OWL-S is an example of a standard that specifies the semantic layer. In this work we will focus on the interfaces of the services and not their statefulness.

It is anticipated that in the future, Grid Services will have associated economical costs with various degrees of quality and performance that can be used for composition and preferred selection of services. Before this higher level information is taken into account, it is important to achieve composition and workflow selection on a fundamental level, where the focus is on the information and state transformation of the services. In this work we represent services by the information transformation they undertake, in the form of input and output parameters which are described by semantics defined in ontologies.

To relieve the user from the unnecessary complexity of specifying exactly which services to use, a semantic description of the information transformation that is needed could be specified in the form of requested capabilities. Based on these requested capabilities a Matchmaker could on-the-fly find the best services and compose them into a workflow. And since semantic interoperability is guaranteed by the semantics, the workflow could be automatically integrated into the client application, which is in our case a Problem Solving Environment (PSE) [7].

We demonstrate in this paper the significance of Semantic Web Services in a PSE and show how a domain ontology is developed to annotate services. Furthermore, we demonstrate composition of services with a simple algorithm and we propose a novel algorithm for automated workflow selection.

The rest of the paper is organized as follows. In Section 2 we give a detailed description of the scenario of the interactive biomedical application. In Section 3 we introduce the architecture of the Matchmaker by describing its interaction with services, clients and ontologies. In Section 4 we discuss semantics for annotating Semantic Web Services, and in Section 5 we develop a domain ontology to annotate the Semantic Grid Services for the biomedical application. Algorithms for automated composition and workflow selection are presented in Sections 4 and 5, and in Section 8 we demonstrate the algorithms by composing and selecting the biomedical services, and finally in Section 9 we conclude.

2 An Interactive Biomedical Application

Problems where blood arteries are weakened or cluttered are called vascular disorders, and lead to reduced or blocked blood circulation, causing in many cases stroke, and eventually death. Medical data acquired by e.g. Magnetic Resonance Angiography (MRA) may be used by specialists to detect these disorders,

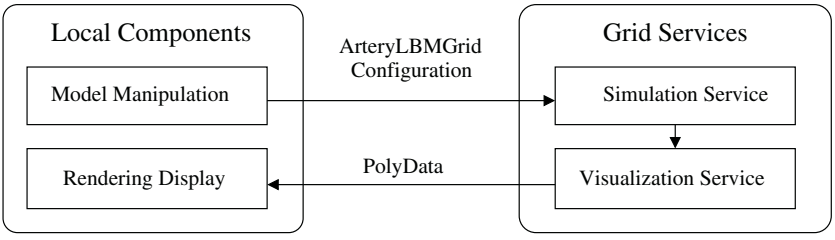


Fig. 1. Dataflow between local components and Grid Services in the PSE

and treatment consists of reconstructing defected arteries or adding bypasses so that the blood flow can normalize. The best treatment is however not obvious. Problem Solving Environments (PSE) are therefore developed to allow surgeons conduct pretreatment planning in a virtual 3D environment [7]. The PSE components and Grid Services for this case study are shown in figure 1. The surgeon works with the Model Manipulation component by inserting or modifying a bypass around the defected part of the blood artery, and when finished submits the data of the artery's geometry (ArteryLBMGrid) to the Blood Flow Simulation Service according to the also provided Configuration.

The Simulation Service produces a dataset (RawVisualizationData) that is sent to a Visualization Service where it is converted into 3D polygons (PolyData), which is then sent to the local rendering component where a visualization of blood flow is displayed to the surgeon. This procedure is repeated until the desired effects are obtained.

The user specifies in the PSE that an information transformation that takes ArteryLBMGrid and Configuration as inputs, and that returns PolyData is needed. This is communicated to the Matchmaker, which tries to find a sequence of services that provides this transformation.

3 Matchmaker Architecture

Figure 2 shows the architecture for the Matchmaker, adapted from [1], depicting the involved parties and their events of interaction. Ontologies are used to annotate capabilities of services that are advertised to the Matchmaker, and then to annotate requested capabilities specified by the client, and based on these requested capabilities the Matchmaker composes and selects the workflows that satisfy the requirements. This involves retrieving the necessary ontologies from the Web, and semantically matching the capabilities using an inference engine. The Matchmaker returns then a proposed workflow of services to the client, that is used to invoke the services.

The Matchmaker maintains the list of advertised services in a local database, and by using the OWL-S API library from The University of Maryland, the capabilities, described by Profile and Process, are parsed for input and output parameters consisting of URIs to ontologies on the Web. The reasoning engine

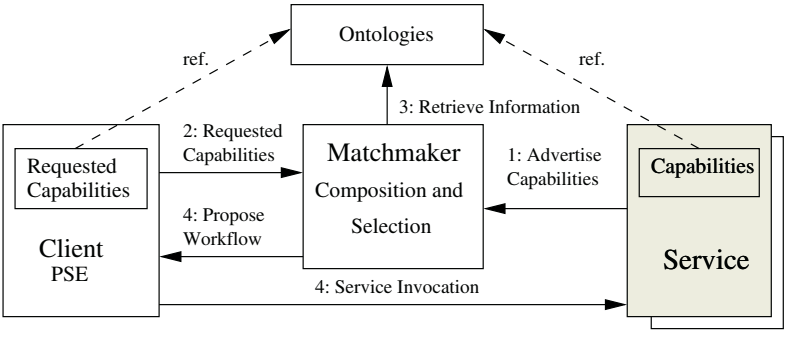


Fig. 2. Architecture with sequence of interaction(solid arrows) and references to ontologies(stippled arrows)

(Racer [12]) loads these ontologies and compares them for equivalence and subsumption.

4 Semantic Data Structures

Input and output parameters of services are represented by semantics defined in domain ontologies, where the ontologies describe the parameters by specifying what information that has to be provided and communicated as part of the parameter. Each parameter points to one concept (core concept) in the ontology, that specifies the necessary information for the parameter, where the information consists of concepts, properties and data values. This is achieved by defining concepts and properties, and then relating the core concept to the concepts and properties through axioms like equivalence and subsumption, specified in OWL as equivalentClass and subclassOf [8], respectively.

Properties, either relating a two concepts or a concept to a data value, are defined with restrictions like allValuesFrom and cardinality, respectively specifying existential qualification of the range-concept and qualified exactly cardinality for the property [8].

The ontologies define the structure that the information has to conform to, and this differs significantly from structure verifications provided by XML Schemas, because the interpretation of the information is based on an ontology that allows complex relations and constraints. Thus, communication with such structures becomes more flexible in that there might be more than one way of satisfying the constraints.

To the best of our knowledge, there exists no adequate name for such semantic structures, thus we coin the name Semantic Data Structure (SDS) to define a group of concepts and relations that are required to satisfy the constraints of a core concept.

A domain ontology may consists of several SDSs with equally many sub concepts and data values. They can therefore grow large and become very complex. Requested and advertised capabilities are annotated with core concepts,

and manually finding these concepts become thus tedious and difficult, and the reusability of ontologies is lowered. Core concepts should therefore be distinguished from other concepts on a semantic level, so that e.g. computer based design tools can be used to select and present core concepts to the user. We propose therefore that core concepts should be subsumed by a certain concept (say CoreConcept) defined in a standardized ontology for Semantic Web Services like OWL-S, thus in this way improving the reusability of domain ontologies. It is furthermore tempting to specify core concepts as inputs and outputs in the domain ontology [9], but outputs for one service might be inputs for another service so that would restrict reusability.

Usage of core concepts can be enforced by only allowing parameters refer to concepts that are subsumed by CoreConcept. For OWL-S, this could be achieved by specifying a range restriction for the property parameterType [3].

5 Domain Ontology

Figure 3 shows fragments of a domain ontology defined to annotate the Grid Services in the biomedical application. The ontology is described with Description Logics notation [11] because of its more compact form than OWL. ConfigLight is a SDS defined as CoreConcept with a property Viscosity, where the data property is restricted to Integer as range, and cardinality of one. ConfigFull is then defined as equal to ConfigLight with ReynoldsNumber as data property. ConfigFull is a specialization of ConfigLight and thus derives property Viscosity and concept CoreConcept. ConfigFull can thus be used in replacement of ConfigLight because the former specifies stricter restrictions and includes more information than the latter. PolyData and PolyDataExtra are defined with similar relation.

$$
\begin{aligned}
&\texttt{ConfigLight} && \equiv \texttt{CoreConcept} \sqcap \forall\ \texttt{Viscosity.Integer} \sqcap \equiv \texttt{1Viscosity}\\
&\texttt{ConfigFull} && \equiv \texttt{ConfigLight} \sqcap \forall\ \texttt{ReynoldsNumber.Integer} \sqcap \equiv \texttt{1ReynoldsNumber}\\
&\texttt{ArteryLBMGrid} && \equiv \texttt{CoreConcept} \sqcap \forall\ \texttt{LocatedAt.Integer} \sqcap \equiv \texttt{1LocatedAt}\\
&\texttt{RawVisData} && \equiv \texttt{RawVisualizationData}\\
&\texttt{PolyData} && \equiv \texttt{CoreConcept} \sqcap \forall\ \texttt{LocatedAt.URI} \sqcap \equiv \texttt{1LocatedAt}\\
&\texttt{PolyDataExtra} && \equiv \texttt{PolyData} \sqcap \forall\ \texttt{NumberOfFrames.Integer} \sqcap \equiv \texttt{1NumberOfFrames}
\end{aligned}
$$

Fig. 3. Fragments of a domain ontology

To demonstrate mapping between concepts, possibly defined in different ontologies, we define RawVisData as equivalent to RawVisualizationData. These core concepts can then be used interchangeable, because RawVisData inherits the property LocatedAt from RawVisualizationData.

SDSs are defined with equivalence axioms and not subsumption. In this way is it sufficient, and not only necessary, to satisfy the concepts and relations on the right-hand side to be a valid member of the SDS. This makes semantic matching more flexible in that different SDSs can refer to the same right-hand side concepts and relations, and thus match as Exact.

6 Automated Composition

In our work, services are composed into workflows based on the information transformation they undertake in the form of input and output parameters. Outputs can either be condition or unconditional, but we are currently for simplicity only considering the latter, thus in this way creating simple workflows that are directly executable without the need for human intervention or machine reasoning during execution.

Service capabilities, requested or advertised, consist of input and output parameters that each refer to a core concept in an ontology. These capabilities are semantically matched as described by Paolucci et al. [10], where valid matching types of SDSs are Exact and PlugIn. Exact match means that the SDSs are interchangeable and equal, while PlugIn match means that an inverse subsumption relationship between the SDSs exists, and which depend whether it applies to input or output parameters. A PlugIn match for an advertised input parameter means that the service requires less information as input then what is asked for or is provided to the service, and for advertised output parameters it means that the service provides more output than what is asked for. PlugIn matches are therefore defined as weaker than Exact matches.

The overall match between two capabilities, either between requested and advertised capabilities, or between capabilities of two succeeding services in a workflow, is determined by the weakest match of the parameters.

Compositions are made by first finding all services with inputs that match with requested inputs, and then in forward chaining fashion, adding services with inputs that match with the outputs of the last service in the workflow. This leads to graphs of workflows where each branch represents a workflow, and composition continues until no more matches among the unused services can be found (each branch holds its own list of unused services), or the outputs of the last service in the workflow match with the requested outputs.

Forward chaining is equally good as backward chaining when conditional outputs are not taken into account, but if they are to be considered then a backward chaining, or goal driven approach, would have been preferred because of the asymmetries introduced by the conditional outputs.

7 Automated Workflow Selection

Several workflows might satisfy the requested capabilities specified in the PSE and the problem is then to autonomously select the most optimal one. We present here an algorithm that evaluates and assigns a cost to each workflow so that they can be compared and thus one workflow selected. Evaluations are based on the types of semantic matches the workflows consist of and the following observations:

- Workflows with as few services as possible are preferred, because services might be located far from each other geographically and communication latency can lower overall performance.

- Exact matches are in general preferred over PlugIn matches, because with services that match as PlugIn there might be information produced that is not used and this might affect the performance negatively.
- Inputs of the first service in the workflow that match as PlugIn with the requested inputs might affect the quality of the outcome differently than PlugIn matches elsewhere in the workflow and should therefore be considered separately. A PlugIn match of the first inputs is considered worse than a PlugIn match in the middle or at the end. This is based on the assumption that what the client requests as input is important and will influence the execution of the workflow. As an example consider a request for a data conversion from one format to another including a certain parameter specifying the conversion. A workflow taking this parameter into account would most likely bring about a conversion closer to what is requested than a workflow not taking this parameter into account.

Equation 1 shows the function that calculates the cost for each workflow, where the inputs and outputs of services are evaluated and given values based on their type of semantic matches. The overall match of the inputs of the first service in the workflow is evaluated and then added to the summation of the matches of the outputs of all N services, where n denotes the service number. Evaluating the input match of the first service separately enables us to specify a matching value uniquely, and for succeeding services in the workflow only the outputs need to be accounted, because an output match is equal to the input match of the next service.

The workflow with the lowest cost is chosen, thus the selection is based on the types of semantic matches and the number of services.

$$Cost_{wf} = Inputs_1(match) + \sum_{n=1}^{N} Outputs_n(match),$$

$$\begin{aligned} Inputs(match) &= \begin{cases} 1 \text{ if match} = \text{Exact;} \\ 2 \text{ if match} = \text{PlugIn.} \end{cases} \\ Outputs(match) &= \begin{cases} 1 \quad \text{if match} = \text{Exact;} \\ 1.5 \text{ if match} = \text{PlugIn.} \end{cases} \end{aligned} \tag{1}$$

8 Demonstration

In the PSE it is specified that a composition is requested that takes ArteryLB-MGrid and ConfigFull as inputs, and that produces PolyData as output. Five different compositions that satisfy these requested capabilities are displayed in figure 4. The workflows consist of two simulation services (FlowSim1 and FlowSim2), two visualization services (Vis1 and Vis2) and one service that simulate and visualize (FlowSimVis).

The inputs of FlowSim1 match as Exact for the requested inputs, while the inputs of FlowSim2 and FlowSimVis match as PlugIn because ConfigLight subsumes ConfigFull (inverse subsumption match). All the matches between simulation and visualization services are Exact, because RawVisualizationData is

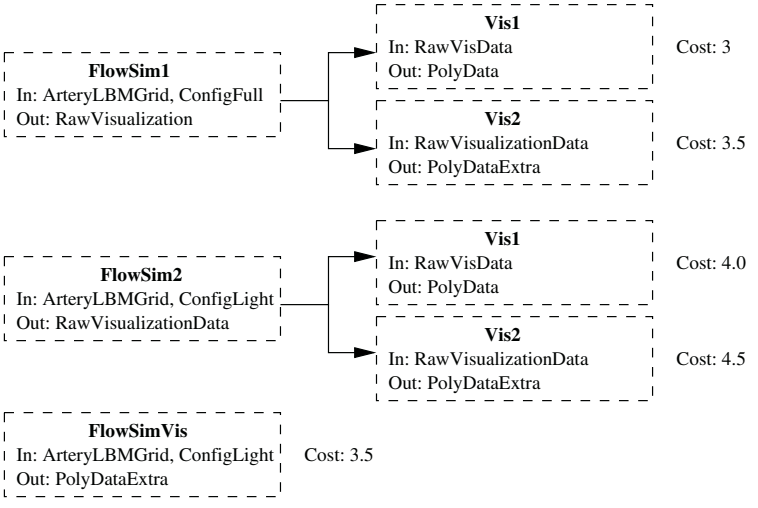


Fig. 4. Workflows with assigned costs

equal to RawVisData in the domain ontology. And the outputs of Vis1 match as exact with the requested outputs, while outputs of Vis2 and FlowSimVis match as PlugIn because PolyData subsumes PolyDataExtra.

The cost of each workflow is calculated with the algorithm in section 3, and the workflow with cost 3 is selected. This workflow is preferred even over the workflow consisting of only FlowSimVis because it brings about a more exact semantic transformation of information.

9 Conclusions and Future Work

This paper identifies a class of applications where Semantic Web Services will be indispensable, in that they enable automated discovery, composition and selection of services for direct integration, without human involvement, into the Problem Solving Environment.

We emphasize that each service input and output parameter refers to a concept in a domain ontology that specifies the semantic description of the parameter. Communicating the information of the parameter require that the concepts and properties defined by this concept is satisfied. We describe this information therefore as a Semantic Data Structure, because it differs from XML Schemas in that interpretation is based on an ontology.

We argue that concepts representing service parameters must be distinguished on a semantic level by sub-classing a certain predefined concept. This enables design tools, and possible agents, to better select the right concepts to annotate requested capabilities, thus improving the reusability of domain ontologies. We propose an extension to OWL-S to enforce such a restriction.

We present an algorithm for automated workflow composition where services are composed into workflows in forward-chaining fashion. The parameters are semantically matched as either Exact or PlugIn, and the overall match between services is determined by the weakest parameter match.

We propose an approach for selecting the best workflow from a set of alternatives, where the quality and cost of each workflow is estimated, based on the types of semantic matches, and number of services, involved.

The algorithms are demonstrated within the context of a biomedical application by composing and selecting Semantic Grid Services that are annotated with a domain ontology.

Future works involves applying the framework to statefull Web Services such as described in the Web Service Resource Framework [5].

References

1. Massimo Paolucci, Katia Sycara, and Takahiro Kawamura. Delivering Semantic Web Services. In Proceedings of the Twelve's World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003, pp 111- 118
2. Grigoris Antoniou and Frank van Harmelen. A Semantic Web Primer. The MIT Press, 2004.
3. Dean, M. (ed). OWL-S: Semantic Markup for Web Services. Version 1.0, 2004.
4. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C, Maguire T., Sandholm, T., Snelling, D., and Vanderbilt, P. Open Grid Services Infrastructure (OGSI) Version 1.0.
5. I. Foster (ed.). Modeling Stateful Resources with Web Services v. 1.1. March 5, 2004.
6. Z. Zhao; G.D. van Albada; A. Tirado-Ramos; K.Z. Zajac and P.M.A. Sloot: ISSStudio: a prototype for a user-friendly tool for designing interactive experiments in Problem Solving Environments, in P.M.A. Sloot; D. Abrahamson; A.V. Bogdanov; J.J. Dongarra; A.Y. Zomaya and Y.E. Gorbachev, editors, Computational Science - ICCS 2003, Melbourne, Australia and St. Petersburg, Russia, Proceedings Part I, in series Lecture Notes in Computer Science, vol. 2657, pp. 679-688. Springer Verlag, June 2003. ISBN 3-540-40194-6
7. P.M.A. Sloot; A. Tirado-Ramos; A.G. Hoekstra and M. Bubak. An Interactive Grid Environment for Non-Invasive Vascular Reconstruction. 2nd Interna-tional Workshop on Biomedical Computations on the Grid (BioGrid'04), in con-junction with Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)
8. OWL Web Ontology Language. W3C Recommendation 10 February 2004.
9. Li, Lei and Horrocks, Ian. A Software Framework for Matchmaking Based on Semantic Web Technology. In Proceedings International WWW Conference, Budapest, Hungary. (2003)
10. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic Matching of Web Services Capabilities. The First International Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.
11. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. The Description Logic Handbook. Cambridge University Press, 2002.
12. Volker Haarsley and Ralf Moller. RACER User's Guide and Reference Manual Version 1.7.7

Grid Application Monitoring and Debugging Using the Mercury Monitoring System

Gábor Gombás, Csaba Attila Marosi, and Zoltán Balaton

MTA SZTAKI Laboratory of Parallel and Distributed Systems,
H-1518 Budapest P.O. Box 63, Hungary
{gombasg, atisu, balaton}@sztaki.hu

Abstract. The Mercury Monitoring System is a generic GMA-compatible grid monitoring framework. In this paper we present how higher level application monitoring and debugging facilities can be built on top of Mercury. Currently two higher level application monitoring systems are implemented using Mercury: one is GRM/PROVE and the other is a stand-alone MPI wrapper library that can be used to learn about the communication patterns of MPI programs. Remote debugging of applications has also been implemented on top of Mercury by using the remote debugging facilities of gdb (the GNU Debugger) which is also presented in this paper.

Keywords: Grid, MPI, performance monitoring, remote debugging.

1 Introduction

As more and more diverse grid environments are becoming available for everyday computation, monitoring and controlling the behaviour of jobs running at remote sites becomes more and more important. Simple sequential applications may be developed, debugged and analysed locally with conventional tools, and even the behaviour of many parallel applications may be analysed on the developer's system using tools like Totalview.

There are cases however where this is not enough. When something does not work as expected it can be difficult to tell whether there is a bug or performance bottleneck inside the application itself or the unexpected behaviour is just an artifact of the internals of the remote system that the user did not know about before. A developer may not have the resources to run a large enough parallel program locally and bugs may arise only when it is submitted to a remote resource to which the developer has no direct access.

To overcome these problems it is essential for a grid environment to provide infrastructure for monitoring and controlling applications. The APART project tried to catalogue all tools being in use today for application performance analysis, their result is available in [2].

The Mercury Monitoring System is being developed as part of the GridLab project [4] as a generic grid monitoring framework. The design goal is to provide

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 193–199, 2005.

support for both resource and application monitoring. In this paper we concentrate on how application developers can benefit from using Mercury by showing two usages: one is for studying communication problems and possible bottlenecks in MPI programs, and the other is support for source-level debugging of applications running on remote grid resources.

2 The Mercury Monitoring System

The Mercury Monitoring System provides a generic GMA-compatible [9] grid monitoring framework. It features a multi-level design where every level acting as a *consumer* of information provided by lower levels may also be a *producer* for higher levels. At the lowest level there is a daemon called the Local Monitor running at every host to be monitored. Local Monitors are grouped together by Main Monitors (usually running at the front-end node of a cluster) that act as data aggregators and policy enforcement points. Consumers may connect to a Main Monitor to request information provided by one or more Local Monitors behind it. See [3] for a complete description of the architecture of Mercury.

Mercury contains two elements to aid application monitoring and steering: an application sensor and an instrumentation library that communicates with this sensor. Together they allow an application to register metrics and controls specific to the application and to receive and serve requests for metrics and controls while the application is running.

The application sensor is loaded into every Local Monitor as a module. It keeps track of processes of jobs running at that node as well as any private metrics or controls that the running applications provide. When the Local Monitor receives a request for a metric or control the application sensor forwards the request to the application process(es) it belongs to. The request is then interpreted by the instrumentation library by performing the measurement or executing the requested operation.

The instrumentation library provides an API for the application developer to connect to the Local Monitor, register application-specific metrics and controls and accept requests for the registered metrics/controls. It communicates with the application sensor using a UNIX domain socket. The instrumentation library also has shared memory support to speed up transferring large volumes of data such as generated by fine-grained instrumentation. The application may also put certain variables into the shared memory area so they can be queried or modified without direct interaction with the application. This is useful for single-threaded applications that do not call the event handler of the instrumentation library for extended periods of time.

Processing of events generated by the application is optimised inside the instrumentation library in the sense that they will not be sent to the Local Monitor if there are no consumers requesting them. This ensures that if an application is built with fine-grained instrumentation it can still run without noticeable performance degradation if the generated events are not requested.

3 Application Monitoring and Debugging

In order to an application to generate performance events, it has to be instrumented. The instrumentation can be either manual or automatic. Manual instrumentation makes it possible to concentrate on events that are really interesting and not to be bothered by unimportant data, which makes evaluation of the collected data very easy and efficient. The drawback of course is the amount of work required to insert the instrumentation calls into the application. Automatic instrumentation relieves the user from this work at the expense that the collected data will likely contain unimportant details making the interpretation harder.

The Mercury Monitoring System provides an API that can be used either directly by the application developer or indirectly by using instrumented libraries. Currently two higher level application monitoring systems are implemented on top of this API: one is GRM/PROVE which is part of P-GRADE [5] and the other is a stand-alone MPI wrapper library that can be used to learn about the communication patterns of MPI programs. Support for monitoring applications using the GAT [4] is also under development.

3.1 P-GRADE

P-GRADE is a development environment for parallel grid applications. Its application monitoring capabilities are described in detail in [1], so it will only be mentioned here briefly.

P-GRADE uses a language called GRAPNEL for describing communication and data flow inside a parallel application at a high level. During code generation GRAPNEL is translated to either PVM or MPI calls depending on the user's choice. The code generator may also insert calls to the GRM instrumentation library around message passing functions if requested by the user. Thus the application will generate trace events before and after a message is sent or received. These trace events may be visualised by PROVE to show when application processes perform useful computation and when are they blocked waiting for communication, as well as computing statistics such as which processes communicate the most.

The GRM instrumentation library can use Mercury to send the trace events to interested consumers. At the user's side the GRM collector connects to the Mercury Main Monitor at the site(s) where the job is running, collects the trace events generated by the application, and feeds them to PROVE generating images as seen on figure 1.

3.2 Performance Monitoring of MPI Applications

Although P-GRADE can do full communication instrumentation and visualisation for parallel programs, it cannot instrument applications that were developed outside of P-GRADE. To overcome this limitation, a stand-alone wrapper library has been recently developed that can be used to monitor the communication of

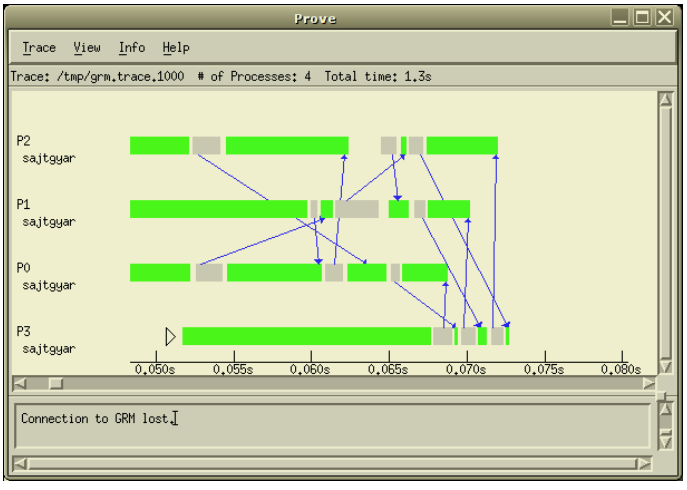


Fig. 1. Visualising MPI communication

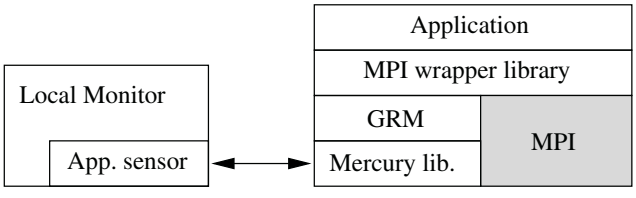


Fig. 2. Using the MPI wrapper library

MPI applications without modifying the source code of the application. Instrumentation is enabled automatically by linking the application with the wrapper library instead of the real MPI library. This solution has the additional benefit that the same wrapper library can be used equally well for applications written in either C or Fortran (or any other language that uses standard MPI library).

The wrapper library was implemented using the MPI profiler interface [7]. The library intercepts calls to a subset of MPI functions and calls the GRM instrumentation library before and after the real MPI call. GRM in turn uses Mercury to send trace events to interested parties (see fig. 2). This way the resulting data can be visualised by PROVE just as if the application were created and started by P-GRADE.

3.3 Remote Debugging

Debugging applications running in a grid environment is often difficult because the user has no direct access to the grid resources. Therefore solutions providing remote debugging facilities are important for application developers.

The OCM-G [6] monitoring system developed by the CrossGrid project supports debugging by providing access to internal variables of the application, allowing the user to manipulate the state of the application and automatically detect certain events (like thread creation). However, utilizing this requires using of tools specific to OCM-G, possibly unfamiliar for grid application developers.

In Mercury, we followed a different approach by building on tools already known to application developers. Remote application debugging has been implemented using Mercury by exploiting the remote debugging facilities of gdb (the GNU Debugger). The original goal of this facility in gdb was to aid debugging applications running on embedded systems over a serial line. Notice that running applications on the grid shows many similarities to embedded systems since usually the owner of a grid job does not have direct access to the systems his or her job is running on.

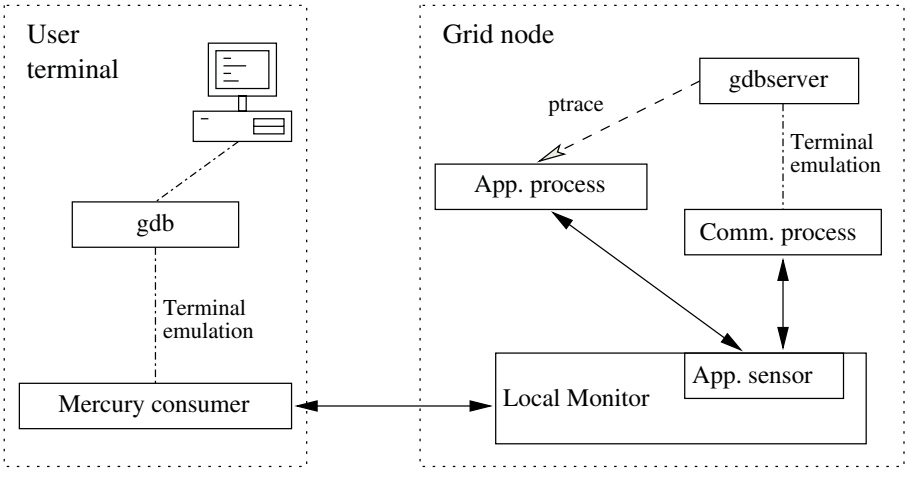


Fig. 3. Remote application debugging

Debugging works as follows. The Mercury application monitoring library embedded in the application connects to the Local Monitor running at the same host. After that the instrumentation library checks for commands from the producer whenever one of the library functions is called; this works best if the application contains fine-grained instrumentation. Alternatively the instrumentation library can export a file descriptor that the application can watch and can call the event handler function in the instrumentation library if there are data available on the file descriptor. This method is useful if the application process already uses asynchronous communication or if the application is multi-threaded.

When the instrumentation library receives a request to execute the control named *application.gdbserver.spawn*, it forks off a communicator process. The communicator process puts itself in the background and creates a virtual master-slave terminal pair. After that the communicator process forks again and starts

gdbserver on the slave side of the virtual terminal, instructing it to attach to the application process. The communicator process talks directly with the Local Monitor instead of using the application's communication channel therefore the application can be debugged even if it detaches from the Local Monitor. After gdbserver has been started, the communicator process simply relays messages between gdbserver and the Local Monitor until either side exits.

At the user's terminal there is a small wrapper application running that fills the gap between *gdb* and Mercury. This wrapper also creates a virtual master-slave terminal pair, forks off a gdb process, and instructs gdb to use the slave side of the virtual terminal for remote debugging as if it were a serial line.

The wrapper application subscribes to the *application.gdbserver.msg* metric at the Mercury Main Monitor that carries the output produced by the remote gdbserver process, and forwards gdb's commands to the remote gdbserver using a control named *application.gdbserver.cmd*. Mercury forwards control invocations to the appropriate Local Monitor(s) which invoke the application sensor implementing the control. The application sensor then sends the data contained in the arguments of the control to the communicator process (and thus gdbserver).

This method for debugging remote applications has several benefits:

- There is no overhead if debugging is not active. The debugger may be started any time, there are no special preparations needed before submitting the job.
- The user may perform the debugging process in the environment he/she is used to without being familiar with the system his/her job is running at. The application can be debugged just as if it were running locally.
- If gdb is built with appropriate cross-debugging support the architecture of the remote host may be different from the user's terminal.
- Since gdb does symbol lookups locally no debugging information has to be present at the remote system. This means the application's binary may be stripped before submitting, usually making it significantly smaller. For developers of proprietary applications this also means that neither the source nor the debugging symbols that might help to reverse-engineer the application have to be given away, yet full source-level debugging is possible remotely.

Debugging works for both sequential and parallel jobs although for parallel jobs a different gdb (and with the current simplistic implementation, a different wrapper process) has to be started for every remote process.

Gdb has good support for building higher level interfaces on top of it, such an interface may be adapted to support grid-level debugging using Mercury with relatively little effort.

The implementation of debugging does not require explicit support from the application so it is possible to put the initialisation of the Mercury instrumentation library in some other library like the MPI wrapper described in the previous section, or into a GAT monitoring adaptor. This means the source code of the application does not have to be modified (although that should not be a limitation if one wants to do source-level debugging).

4 Conclusion

The Mercury Monitoring System is a generic grid monitoring framework. It is being used in the European GridLab project and the Hungarian SuperGrid project, and is about to be deployed in the Hungarian ClusterGrid project. In this paper we described how higher level application monitoring and remote debugging was implemented on top of the generic framework. The flexibility and modularity of the Mercury architecture and using gdb made implementing the remote debugging support easy. Our choice of gdb provided the additional benefit that the developer can use an already familiar user interface.

The remote debugging support is part of the latest Mercury release. Tracing of communication of parallel programs using Mercury is already part of the newly released P-GRADE 8.3 version while the stand-alone MPI wrapper library is available separately. Support for tracing applications using the Grid Application Toolkit is under development.

Acknowledgements

This work was sponsored by the European Commission under contract number IST-2001-32133 and the Hungarian Scientific Research Fund (OTKA) under grant number T042459.

References

1. Z. Balaton, P. Kacsuk, N. Podhorszki, F. Vajda: From Cluster monitoring to Grid Monitoring Based on GRM. Parallel Processing: proceedings / Euro-Par 2001., pp. 874-881
2. M. Gerndt, R. Wismüller, Z. Balaton, G. Gombás, P. Kacsuk, Zs. Németh, N. Podhorszki, H-L. Truong, T. Fahringer, M. Bubak, E. Laure, T. Margalef: Performance Tools for the Grid: State of the Art and Future. APART White Paper
3. G. Gombás, Z. Balaton: A Flexible Multi-level Grid Monitoring Architecture. F. Fernandez Rivera et al. (Eds.): Across Grids 2003, LNCS 2970, pp. 257-264.
4. The GridLab Project. `http://www.gridlab.org`
5. P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas, N. Podhorszki, Z. Balaton, G. Gombás: P-GRADE: a Grid Programming Environment. Journal of Grid Computing, Volume 1, Issue 2, 2003, Pages 171-197.
6. B. Baliś, M. Bubak, M. Radeczki, T. Szepieniec, R. Wismüller: Application Monitoring in CrossGrid and Other Grid Projects. Marios D. Dikaiakos et. al. (Eds.): Across Grids 2004, LNCS 3165.
7. MPI - The Message Passing Interface standard. `http://www-unix.mcs.anl.gov/mpi`
8. R. Ribler, J. Vetter, H. Simitci, D. Reed: Autopilot: Adaptive Control of Distributed Applications. Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, July 1998.
9. B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, R. Wolski: A Grid Monitoring Architecture. `http://www.gridforum.org/Documents/GFD/GFD-I.7.pdf`

Interactive Visualization of Grid Monitoring Data on Multiple Client Platforms

Lea Skorin-Kapov[1], Igor Pandžić[2], Maja Matijašević[2], Hrvoje Komerički[2], and Miran Mošmondor[1]

[1] Research and Development Center, Ericsson Nikola Tesla, Krapinska 45, HR-10000 Zagreb, Croatia
{lea.skorin-kapov, miran.mosmondor}@ericsson.com
[2] FER, University of Zagreb, Unska 3, HR-10000 Zagreb, Croatia
{igor.pandzic, maja.matijasevic, hrvoje.komericki}@fer.hr

Abstract. Most current Grid monitoring systems provide a visual user interface. With recent advances in multimedia capabilities in user terminals, there is a strong trend towards interactive, multi-modal and multi-platform visualization. In this paper we describe a multi-platform visualization architecture and a Web based service built upon it, which provides a view of the monitored Grid hierarchy, and the values of selected monitoring parameters for different Grid sites. We demonstrate the application on four platforms: a desktop Personal Computer (PC), a handheld PC, a Java-enabled new-generation mobile phone, and a Wireless Application Protocol (WAP) enabled mobile phone.

1 Introduction

Grid technologies have been described as supporting the sharing and coordinated use of diverse resources in distributed "virtual organizations" [1]. As Grid structure is inherently difficult to monitor and manage due to its many geographically and organizationally distributed components, tools and services to assist humans in such tasks have become a necessity. To date, a number of Grid monitoring systems has been developed to enable monitoring and displaying Grid topology, its components, and their selected parameters, for example, resource availability, load distribution and task performance [2][3][4][5]. In addition to data collection, most current systems also include a means for Web-based visualization of the monitoring information gathered by the system.

With recent advances in multimedia capabilities in user terminals, there is a strong trend towards improved interactivity as well as rich media spaces for information visualization [6]. Visualizations are becoming *multi-modal* and *multi-platform*, i.e. they may combine *various media* such as text, hypertext, pictures, multi-dimensional graphics, audio, and video, on a *wide range of client (end-user) platforms*, from PCs to new-generation mobile phones.

In this paper we describe a multi-platform visualization architecture and a Web based service built upon it, which provides a user with a view of the monitored Grid hierarchy, and the values of selected monitoring parameters for different Grid sites.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 200 – 210, 2005.

The data source used by our prototype was a central data repository provided by the MonALISA system [4], however, the described architecture is independent of data source, and the particular implementation described in this paper could be tailored to work with a different database or data source (e.g., a publish/subscribe system) by use of Web services. We also believe it to be suitable for integrated monitoring systems [7], where data could come from different sources but could be presented through a common visual interface. The prototype service we implemented provides an interface for users to view Grid configuration and monitoring data, such as load, data rates, and memory on different platforms. The capability for multiplatform visualization and flexibility in introducing new visualization techniques make our approach a viable alternative to custom-made graphical user interfaces.

The paper is organized as follows. First, we briefly analyze the properties of typical client platforms with their communication and presentation capabilities. Next, we describe the proposed architecture and its implementation. Finally, we demonstrate the application of the proposed architecture on Grid monitoring data visualization on four platforms: a desktop Personal Computer (PC), a handheld PC, a Java-enabled new-generation mobile phone, and a Wireless Application Protocol (WAP) enabled mobile phone.

2 Client Platform Capabilities

With the growing heterogeneity arising from differences in client devices and network connections there is a need to adapt services to device networking and presentation capabilities. Various parameters dictate client platform capabilities (e.g. available memory, processor speed, graphics card, display) and connection types (e.g. LAN, Wireless LAN, General Packet Radio Service (GPRS)). Table 1 gives an overview of some client devices that were used in this work as test platforms, with their respective characteristics, processing capabilities, and communication capabilities.

For the purposes of this paper we divide client platforms with regards to visualization capabilities into three groups (full, midi, mini) as follows:

Full clients: platforms with enough processing power and visualization capabilities to locally process the raw data received from a server and visualize it while simultaneously supporting all visualization modes (2D and 3D graphics, tables, text, etc). In addition, the full client may offer better interaction capabilities with the data. Although other types of full clients may emerge, the representative of a full client is a PC (Windows, Linux, etc.) running a standard Web browser with Java support. Example interfaces and visualization tools may be implemented as Java applets (e.g. Shout3D) and standard Hypertext Markup Language (HTML) elements, so there is no need to download/install any software on the client machine. Additional software, however, may be installed if needed depending on the interface implementation (e.g. Virtual Reality Modeling Language (VRML) [8] plug-in for 3D graphics). Client hardware and software may vary from lower-end configurations to higher-end configurations.

Midi clients: platforms with medium processing power and visualization capabilities, powerful enough to present a user interface, simple 2D/3D graphics, as well as text

and tables. The representative of a midi client would be a Personal Digital Assistant (PDA) (e.g. Compaq iPAQ or Palm) or a PDA-integrated mobile phone (e.g. Sony Ericsson P800).

Mini clients: platforms that have insufficient processing power for local processing of raw data, and insufficient presentation capabilities for showing either 2D or 3D graphics. Such a terminal would receive pre-formatted Wireless Markup Language (WML) pages ready for presentation instead of raw data.

Table 1. Client platforms – processing and communication capabilities

		Full Client PC (low-end)	Full Client PC (high-end)	Midi Client PDA	Midi Client Smart Phone	Mini Client Mobile phone
General Characteristics	Terminal example	Desktop PC low-end	Desktop PC high-end	Compaq iPAQ 3870	Sony Ericsson P800	Ericsson R520s
	Operating system	Windows 2000/XP	Windows 2000/XP	Windows Pocket PC (CE 3.0)	Symbian 7.0	proprietary
	Browser	IE 5.5 / Netscape 6.0	IE 6.0 / Netscape 7.1	IE 3.02	Opera 6.0 / SE R101	EricssonR520 R201 WML browser
		HTML, VRML, Shout3D	HTML, VRML, Shout3D	HTML, VRML, Shout3D	HTML WML	WML
Processing Capabilities	Processor	PIII 800 MHz	P4 2.66 GHz	Intel Strong ARM SA 1110 206 MHz	ARM 9 156 MHz	N/A
	Speed	1066 MIPS	4904 MIPS	235 MIPS	N/A	N/A
	Memory size	128 MB	512 MB	64 MB	16+16 MB	N/A
	Display size	1024x768	1280x1024	240x320	208x320	101x65
	Color depth	32-bit	32-bit	12-bit	12-bit	1-bit
	Sound	16-bit	16-bit	8-bit	16-bit	none
		Stereo	Stereo	Mono/speaker, Stereo/headph.	Mono/speaker, Stereo/headph.	none
Communication Capabilities	Network connec-tion	LAN 100 Mbps	LAN 100 Mbps	GPRS CS-2 53.6 kbps / WLAN 11 Mbps	GPRS CS-2 53.6 kbps	GSM 9.6 kbps / GPRS 48 kbps
	Latency	<10 ms	<10 ms	1 s (GPRS)	1 s	0.5 / 1 s
	Jitter	<1 ms	<1 ms	N/A	N/A	N/A
	Packet loss	<1%	<1%	N/A	N/A	N/A
	BER	$<10^{-8}$	$<10^{-8}$	10^{-3} / 10^{-5}	10^{-5}	10^{-3} / 10^{-5}

In addition to service customization based on client processing and communication capabilities, customization may be introduced through user preferences. For example, a user may wish to filter some media types (e.g. sound, video, etc.) in order to

increase response time from the system and thus increase the functionality of a specific application interface at the expense of richness of presentation detail.

3 Multiplatform Universal Visualization Architecture

The architecture we apply to for visualization of Grid monitoring data is the Multiplatform universal visualization architecture (MUVA) [9]. MUVA provides universal visual access to data independent of the client platform, while automatically adapting delivery modes to the particular platform. Rather than developing data visualization applications designed to run on a target (group of) client platforms, we separate the platform adaptation procedure on the output side from the implemented data visualization technique, and so facilitate flexible client access. In addition, by separating the data source on the input side from the visualization technique, the result is reusability of such techniques across a wide range of application domains and data sources. For example, the implementation of a technique such as a 3D tree representation of a hierarchical structure may be reused when developing any application which makes use of hierarchical data visualization.

MUVA has been designed as a flexible and modular architecture comprised of a collection of software modules. Fig. 1 presents a conceptual view of the MUVA architecture. Crucial parts of the architecture are the *visualization tools*, which represent various modes and concepts of visualizing the data (e.g. text, table; graph; chart; tree). Visualization tools are separated from actual client devices by *platform drivers*, designed to adapt the data delivery mode to specific platforms. On the input side, actual data collection is separated from the abstract visualization tools. This allows for any data source to be connected simply by developing thin *application interfaces*. The result is quick adaptability to various specific application domains.

The *service logic* provides the necessary intelligence for connecting application interfaces, visualization tools, and platform drivers depending on the particular client platform capabilities and user request. Each component of the architecture contains several modules, where not all of them have to be used in each application. The modular design and separation of MUVA components allows for easy addition, modification, and maintenance of software modules. A more detailed description of MUVA components is given below.

Visualization tools are responsible for one particular mode of visualization (tool), e.g., a 3D structure of an input hierarchy, a simple table, bar-chart, pie-chart, etc. Each tool is *standardized*, in terms of (1) input data parameters that can be fed to it through its API; (2) requests it can receive through its request interface; and 3) visual output it produces in reply to a given request.

Platform drivers are implemented for each supported platform. They render (visualize) formatted data received from visualization tools on the screen, and enable user interaction. The communication with the visualization tools may be through the network or local, depending on the location of the visualization tool in relation to the platform driver. Any or both may in certain cases be located on the server side, and in other cases on the client.

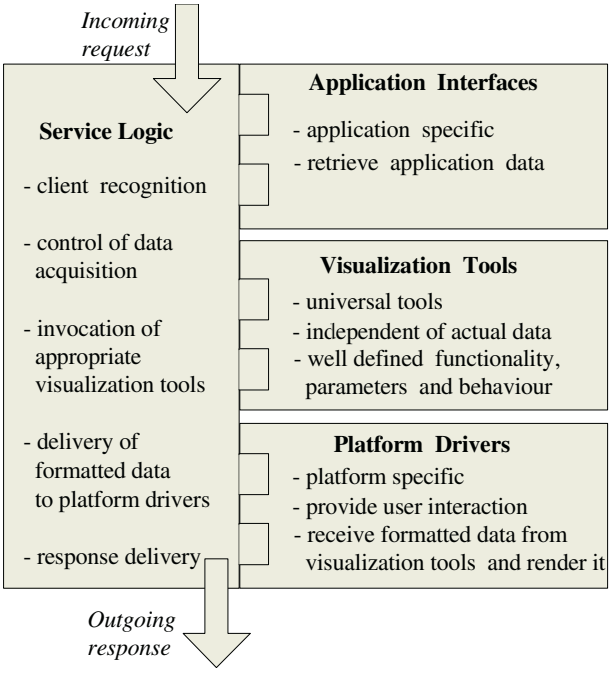


Fig. 1. MUVA concept

Application interfaces are responsible for retrieving data from a data source via a standard application specific API. Retrieved data is converted to Extensible Markup Language (XML) format based on the specified input interface for visualization tools.

Service logic encompasses modules that provide the intelligence needed to bring together the components of the architecture in order to enable universal visual access and delivery mode adaptation.

To illustrate the typical usage scenario, we start from the incoming client request. Upon receiving a client service request, the client's preferences and platform capabilities are identified. One method of identification is based W3C Composite Capabilities / Preference Profile (CC/PP) Recommendation for device independence [10], a proposed industry standard for describing delivery context. The client profile data format is based on the Resource Description Framework (RDF). A client profile may either be sent directly as an extension to a HyperText Transfer Protocol (HTTP) request, or referenced from a remote location using a Uniform Resource Locator (URL). The implementation of client identification based on a set of generic profile parameters allows for on-the-fly identification of the capabilities of an increasing number of end-user devices.

The service logic retrieves raw data independently of the platform capabilities through invocation of application interface modules. The raw data is then sent to appropriate visualization tools. Formatted data received as the output from visualization tools is then delivered to necessary platform drivers. The service logic

layer provides the logic necessary to select adequate visualization tools and platform drivers to produce the final content, adapted to the given client platform.

In order to demonstrate the proposed approach in a real world scenario, a prototype Web based service was implemented. In the following section, we describe a service providing multiplatform visual access to Grid monitoring data. Service implementation helps to demonstrate the separation between application interfaces, visualization tools, platform drivers, and service logic components, as well as the main communication channels involved.

4 Implementation

The service implemented in this work provides an interface for users to view network configuration and monitoring data, such as load, data rates, and memory on different platforms. As mentioned earlier, the data source used was a central data repository provided by the MonALISA system, which provides a distributed monitoring service and was in this case used to monitor hundreds of AliEn Grid sites (http://alien.cern.ch/).

Users access the service by entering a unique URL, which is independent of the client device being used. Requested data, which is then retrieved from a central repository and described using XML, is dynamically converted to a format suitable for displaying on the client device. The different formats that were used include VRML, HTML, and WML. The service implementation was tested on four platforms: a desktop PC, a handheld PC, a Java-enabled PDA-type mobile phone, and a standard WAP-enabled mobile phone.

Where possible, the monitored network configuration (or a particular sub-configuration) was visualized using the 3D Cone Tree technique [11]. Cone Tree is an interactive visualization technique suitable for hierarchical structures. The root of the network hierarchy is located at the tip of a transparent cone. When a level in the hierarchy is expanded (on user click), its children nodes are distributed at equal distances around the base of a cone. The user interface is enhanced by enabling interactive viewing, zooming, expanding and collapsing of parts of the structure. A formal user study using a cone-tree-based file system visualization showed that although Cone Trees are not suitable for all tasks, users “were enthusiastic about the cone tree visualization and felt it provided a better ‘feel’ for the structure of the information space” [12].

We now describe the implementation of the MUVA system architecture components.

Service logic – Client recognition functionality was implemented using Apache server 2.0.47. Due to fact that the developed prototype service was intended to be made available to users outside of a laboratory environment, and the current lack of widespread availability of CC/PP compliant terminal browsers, implemented client recognition is in this case based simply on identification of the browser type specified in the User-Agent header field of the HTTP request.

Data is retrieved in standard XML format through invocation of application interface components and formatted depending on the platform capabilities and requests of the client. Requests are directed towards a Java servlet that is run using

Apache Tomcat Server 4.1.27-LE-jdk14. The servlet requests data, and invokes the necessary visualization tool. In cases when 3D content is generated, the Cone Tree tool is called. Once the VRML result is received, the servlet calls on a platform driver to further adapt the VRML file for rendering in a Shout3D (Eyematic Interfaces Inc., www.shout3d.com) applet, after which the HTTP response is sent to the client.

In cases where the content generation is based on HTML or WML format, the servlet passes retrieved XML data to Apache Cocoon 2.0.4. Data is formatted into HTML tables or histograms using Extensible Stylesheet Language Transformations (XSLT) technology. Where necessary, additional platform drivers are invoked to further adapt the format, prior to sending the response to the client.

Application interfaces – Data is always requested by application interfaces and returned in standard XML format. The interface towards the actual data repository storing monitoring data collected by the MonALISA system is based on Web Service technology. Connectivity to the Web Service was provided using Apache Axis 1.1 open source solution, the follow up on the Apache SOAP project. The stub code for the Web Service was generated by Axis' WSDL2Java utility and modified according to our needs. The Web Service returns values in the form of Java beans, that are then transformed to XML format.

Visualization tools – Once data is retrieved, visualization tools are needed to generate the actual data representation. Various visualization techniques were used, including text, 2D graphics, and 3D graphics.

The creation of a VRML Cone Tree display based on an input hierarchy was implemented using Java. In general, any form of hierarchical data structure may be given as a valid input. Optional additional parameters may be applied to create a simpler Cone Tree display (e.g. when the client device is a PDA), where certain elements of the tree hierarchy are filtered (e.g. display only nodes or clusters belonging to a particular farm). If a particular terminal is not capable of displaying or rendering complex 3D graphics, data is transformed to simple HTML tables or histograms by using XSLT.

Platform drivers – The interface displaying the 3D scene, designed to be viewed on a client with a standard Web browser and Java support, was implemented as a Java applet and based on the Shout3D engine. Shout3D is a library of Java classes for rendering 3D scenes over the Internet, thus offering the user the ability to view and interact with 3D scenes without the need for any additional plug-ins. In a different set-up, Cortona VRML plug-in and Pocket Cortona (for rendering on the iPAQ PDA) were used to render the 3D scene. Within the scope of MUVA, the Shout3D applet classes and Cortona plug-in are all considered platform drivers.

In addition to displaying 3D content on the PC and iPAQ clients, we implemented a C++ application to dynamically generate a 3D scene (in our case a 3D Cone tree display) on the Sony Ericsson P800 mobile device. The DieselEngine SDK 1.3 was used for software support. It is a collection of C++ libraries for creating 3D applications on mobile devices. Additional software used included Symbian UIQ v7.0 SDK and Metroworks CodeWarrior for Symbian OS. The application that was built reads a VRML file dynamically generated by the Cone tree visualization tool, parses the file, converts it to Diesel3D scene format and displays the content to the user. Additional interaction modules for navigation, camera manipulation, and object

selection within the scene were also implemented. Also, XSLT files were implemented to further adapt content for display on a particular device. This includes creation of WML format for display on a WAP-enabled mobile phone.

5 Results

The result is a multiplatform Grid data visualization service that is developed in a modular fashion in order to facilitate adaptation across different application domains. The implemented service provides visualization of monitoring parameters for a large number of Grid nodes. The monitored nodes are arranged in a hierarchical manner into farms and clusters, referring to the geographical and/or logical grouping of nodes into virtual computing systems. The MonALISA system collects monitoring data from all distributed sites and stores the data in a central repository. Data is collected by our service from the central repository in CERN via a Web Service interface.

A view of the *full client* display interface is shown in Fig. 2. Upon initial loading, the *3D view window* renders the 3D scene displaying the dynamic node configuration. The *Parameters window* enables a user to choose a monitoring parameter. The *Histogram window* enables a user to choose between displaying real-time data and history data. Once the user has chosen a parameter and histogram button, clicking on the "Execute!" button will initiate the coloring of tree nodes and writing text to the output window. Parameter values are retrieved for each node, and coloring is based on the range that the value fits into. These ranges are displayed in the *Legend window.*

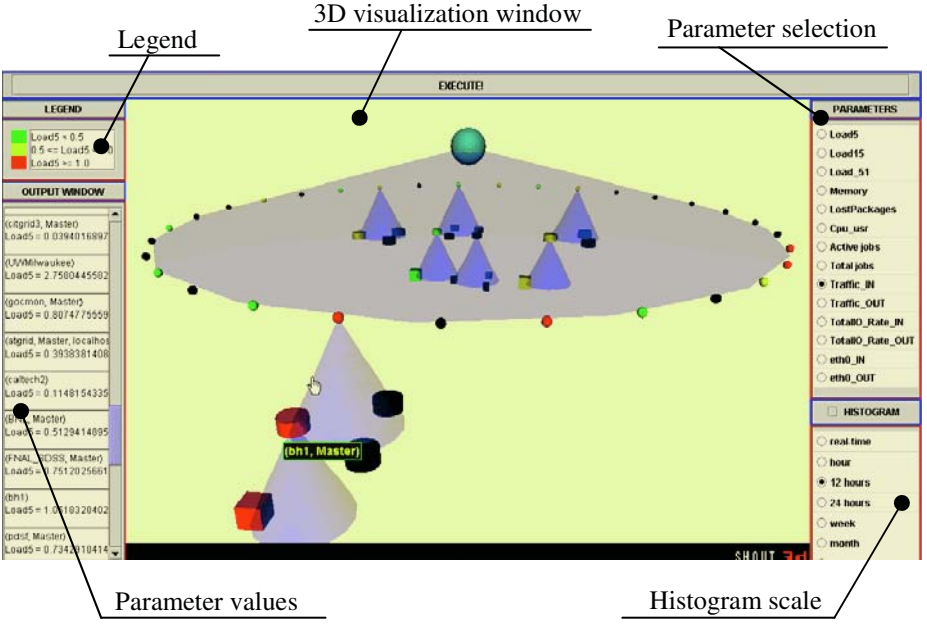


Fig. 2. Visualization using a *full client*: 3D visualization and text windows

Fig. 3. Visualization using a *midi client* (iPAQ)

Fig. 4. Visualization using a *mini client* (Sony Ericsson P800)

Fig. 5. Visualization using a *mini client* (WAP enabled mobile phone)

When a user accesses the same service using an iPAQ (Fig. 3), the same data is represented using a combination of HTML pages (to specify the network

configuration) and simpler 3D Cone trees with only subsets of nodes shown (e.g. a user chooses to view monitoring parameter values for a specific cluster). In cases where a large hierarchy needs to be presented, small display size and lower processing capabilities make it more efficient to present the data using a simpler method, such as a table that a user can scroll through.

The visualization is also displayed on the Sony Ericsson P800 mobile device, where data is again represented using a combination of HTML pages and 3D content (Fig. 4).

If a user using a WAP enabled mobile phone accesses the service, the result is data presented in WML format. Simple WML pages allow the user to list through the network configuration, and request monitoring parameter values (Fig. 5).

6 Conclusion

In this paper, we have presented the concept of a multiplatform universal visualization architecture and its application to visualization of Grid monitoring data on multiple platforms. The key features of the proposed approach are independence of data acquisition and the thin adaptation "layer" for the platform on which the data is visualized. The presented case study demonstrates the visualization of Grid monitoring data on multiple platforms: a desktop PC, a handheld PC, and various mobile phones. Our future work is directed towards extending the system with the ability to interact not only with the display, but also with the Grid itself, as well as towards introducing new visualization tools, suitable for mobile devices.

References

1. Foster, I., C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers (1998)
2. Andreozzi S., N. De Bortoli, S. Fantinel, A. Ghiselli, G. Tortone, C. Vistoli. GridICE: a monitoring service for the Grid, Proceedings of the Third Cracow Grid Workshop, Cracow, Poland (2003)
3. Balaton, Z., P. Kacsuk, N. Podhorszki, F. Vajda, Comparison of Representative Grid Monitoring Tools, Laboratory of Parallel and Distributed Systems, Computer and Automation Research Institute of the Hungarian Academy of Sciences, Technical Report LPDS-2/2000 (2000)
   [Available: http://www.lpds.sztaki.hu/publications/reports/lpds-2-2000.pdf]
4. Newman, H. B., I. C. Legrand, P. Galvez, R. Voicu, C. Cirstoiu. MonALISA: A Distributed Monitoring Services Architecture. Proceedings of 2003 Conference for Computing in High Energy Nuclear Physics, La Jolla, California, USA (2003)
5. R-GMA: Relational Grid Monitoring Architecture, http://www.r-gma.org
6. Card, S. K., J. D. Mackinlay, B. Shneiderman. Readings in information visualization: using vision to think. Morgan Kaufmann Publishers (1999)
7. Mambelli, M., R. Gardner. Integration of Monitoring Systems for Grid Environments, Proceedings of the 13th IEEE International Workshop on Enabling Technologies WET-ICE: Infrastructure for Collaborative Enterprises (2004) 266-267

8. Information Technology – Computer graphics and image processing – The Virtual Reality Modeling Language (VRML) – Part 1: Functional specification and UTF-8 encoding. ISO/IEC 14772-1:1997 (1997)
9. Skorin-Kapov, L., D. Mikic, H. Komericki, M. Matijasevic, Pandzic, I. Multiplatform Universal Visualization Architecture, Proceedings of the Second International Conference on Advances in Mobile Multimedia 2004, Bali, Indonesia (2004)
10. Butler, M., F. Giannetti, R. Gimson, T. Wiley. Device Independence and the Web. IEEE Intenet Computing 6, 5 (2002) 81–86
11. Robertson, G. G., J. D. Mackinlay, S. K. Card. Cone trees: Animated 3D visualization of hierarchical information. Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology, ACM Press, New York, USA. (1991) 189–184.
12. Cockburn A., B. McKenzie. An Evaluation of Cone Trees, In People and Computers XIV: Proceedings of the HCI 2000, 14th Annual Conference of the British Human Computer Interaction Group (2000) 425–436

GridBench: A Workbench for Grid Benchmarking

George Tsouloupas and Marios D. Dikaiakos

Department of Computer Science,
University of Cyprus, 1678 Nicosia, Cyprus
{georget, mdd}@ucy.ac.cy

Abstract. In this article we present the GridBench, an extensible tool for benchmarking and testing Grid resources. We give an overview of the GridBench services and tools that provide easy invocation of benchmarks and management of results. We also show how the tool can be used in the analysis of results and how the measurements can be used to complement the information provided by Grid information services and used as a basis for resource selection. In order to illustrate the usage of the tool, we describe scenarios for using the GridBench framework and the GridBench "virtual workbench" to perform benchmarking experiments and analyze the results.

1 Introduction

High Performance Computing and it's users have greatly benefited from benchmarking over the years; benchmarking can be just as beneficial for computational Grid computing. Benchmarking metrics published on the Grid can provide a basis for users to assess the "quality of service" expected of a Grid resource or a Virtual Organization providing computational services at a given cost. Grid benchmarks can be used by middleware developers to compare different middleware solutions such as job submission services, resource allocation policies, scheduling algorithms, etc. Grid-oriented benchmarks can serve as an evaluation of the fitness of a collection of distributed resources for running a specific application. As common programming models or paradigms start to emerge for programming in Grid environments, Grid benchmarks can serve as a feasibility study of running a general class of applications (or applications following a similar programming paradigm). A key aspect of Grids and Grid resources is their dynamic nature and Grid benchmarks can help study the effect of this dynamic nature of the Grid on application performance. Additionally, they can provide some insight to the properties of Grid Architectures.

The heterogeneity of Grid platforms and the dynamic nature of Grid resources makes the archival and interpretation of measured metrics a complex task and raises questions about the overall applicability of benchmarking. Existing platforms are largely under continuous re-design and development, with very limited cross-platform interoperability, making the specification, submission, and management of jobs is a tedious process. Measuring and/or monitoring

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 211–225, 2005.

performance metrics at the application level of the Grid is currently a target of ongoing research work. Performance measurements are affected by a variety of factors, including the characteristics of resources allocated for a particular run, the time-dependent latency and bandwidth of shared Internet links used for communication between remote sites, the performance capacity of middleware libraries used at the application level, etc.

In the remainder of this article we describe the GridBench tool for benchmarking and testing Grids. In the next section we describe our current implementation of the Gridbench architecture, services as well as the GridBench User Interface and how we used it to perform experiments on a mid-sized Grid infrastructure. Finally we provide some use-case scenarios and results.

2 GridBench

Grids and Grid Resources in general are characterized by static information provided by Grid Information systems. Grid end users and central Grid Services (such as resource brokers) need a better source of information on which to base decisions. These decisions mainly refer to resource allocation or scheduling decisions. The use of results from micro and macro-benchmarks can improve the decision making process, but at this point there is no easy, automated way to obtain, manage and deliver these measurements. GridBench is aimed at fulfilling this purpose.

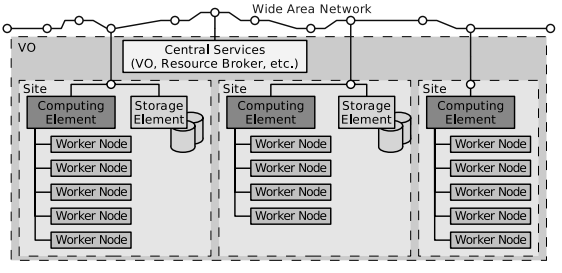


Fig. 1. A generic view of the target infrastructure

GridBench assumes an underlying hardware infrastructure that loosely adheres to the one depicted in figure 1. This basic infrastructure, a Grid Virtual Organization (VO) consists of a set of geographically distributed sites connected over a shared network (i.e. the Internet). Each site contains a Computing Element which manages a set of "Worker Nodes" for performing computations. Typically a CE is associated with a "Storage Element", which is an interface to mass storage, and to which it has direct (Local Area Network) access (e.g. via the Network File System). The Grid VO also contains some VO services such as Grid Information Services, a resource broker, VO membership server etc.

GridBench, as a tool for benchmarking grids, has two main objectives:

1. Generate metrics that characterize the performance capacity of resources belonging to a Virtual Organization and *spanning across multiple Grid nodes*, in terms of computational power, file-transfer speed, inter-process communication bandwidth, application-kernel performance, scalability etc.
2. Provide a *tool* for researchers that wish to investigate various aspects of Grid performance, using well-understood kernels that are representative of more complex applications deployed on the Grid. Having access to a corpus of such kernels and being able to easily specify and dispatch parameterized runs of these kernels on Grids, facilitates the characterization of factors that affect application and infrastructure performance, the quantitative comparison of different middleware solutions, algorithms for scheduling, resource allocation, etc.

To address the two main objectives mentioned, Gridbench has two constituents: the *GridBench Benchmark Suite* and the *GridBench Benchmarking Framework*. The GridBench Benchmark suite is a collection of new and existing micro-benchmarks, micro-kernel benchmarks and application benchmarks; its purpose is to generate the metrics that will characterize resources and virtual organizations. The GridBench suite takes a layered approach as shown in figure 2. The multi-layered structure of the Grid (shown in figure 1) calls for performance measurements at the different layers of the Grid. GridBench seeks to investigate performance properties of the following "layers" of the Grid architecture:

1. The Resource, for example a cluster node or a Storage Element;
2. The Site, which is a collection of resources interconnected through a local- or system-area network, and belonging to one administrative domain(e.g. a cluster of PCs or a symmetric multiprocessor system);
3. The Grid Constellation, which includes multiple sites constituting the computing platform of a Virtual Organization.
4. The Middleware, that is the software layer providing access to shared resources of a Grid constellation and which gives the programmer the Grid as a shared resource.

The suite includes benchmarks for CPU (Floating Point and Integer operations), memory bandwidth, cache performance, detecting available physical memory size, interconnect performance (MPI), synthetic benchmarks and application kernels. A detailed description of the GridBench suite is beyond the scope of this article, more details on the Gridbench suite can be found in [10, 11].

2.1 The GridBench Back-End

The GridBench Benchmarking Framework provides facilities for defining and running benchmarks as well as archiving, retrieving and analyzing the results of the GridBench benchmark suite.

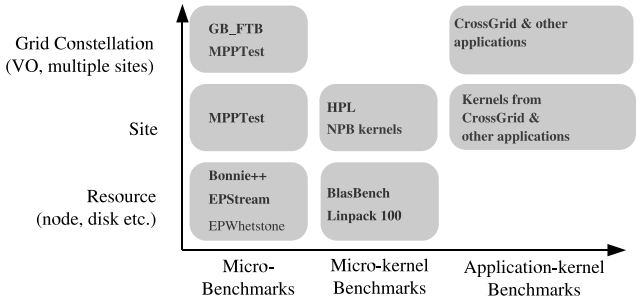


Fig. 2. A layered approach to benchmarking, with *micro-benchmarks*, *micro-kernel* benchmarks and *application benchmarks* on the x-axis, and *resource*, *site* and *grid constellation* on the y-axis

GridBench was designed to be as independent of specific middleware as possible. The design is open enough to allow easy replacement of the underlying middleware by the use of *Middleware plugins*. Currently implemented are plugins for Globus and the LCG2-compatible [8] EU CrossGrid middleware. The user can use the Globus MDS [5] for information retrieval, and either the EU CrossGrid [6, 7] Resource Broker or the Globus GRAM for job execution.

2.2 Overview

Figure 3 outlines the software architecture of GridBench and (at a very high level) indicates which components interact with each other. This is indicated by a line connecting the two interacting components. The main components of this architecture are:

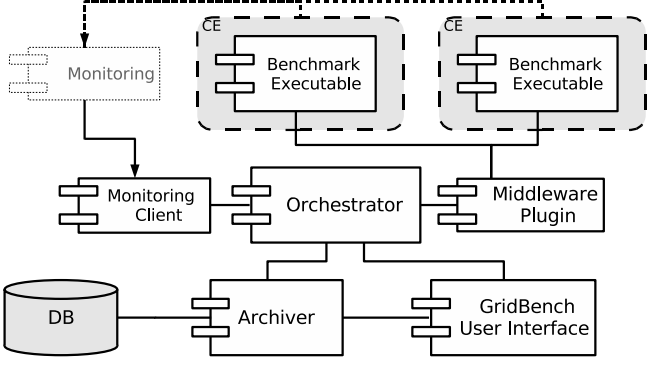


Fig. 3. The GridBench architecture overview. An outline of system's major components and their interaction

- the **GridBench Suite**
  - is made up of the benchmark executables (e.g. Linpack).
- the **GridBench UI**
  - Interacts with the *Archiver* to retrieve benchmark *models*[1] and results.
  - Defines and submits benchmarks to the *Orchestrator*.
  - Analyzes results, create charts.
- the **Orchestrator** web-service
  - Accepts benchmark definitions generated by the *GridBench GUI* (or any other source) and manages their execution by the use of the appropriate *Middleware Plugin*.
  - Monitors the job status for each benchmark job and on completion retrieves and archives the resulting metrics.
- the **Archiver** web-service;
  - Maintains a repository of benchmark results and model definitions.
  - Provides an interface to a relational database back-end.
- the **Middleware Plugin**
  - Middleware-specific job execution, output retrieval.
  - Translation of XML descriptions of benchmarks to a job description language;
  - There are currently two implementations of the Middleware Plugin interface: one for Globus and one for the EU-CrossGrid middleware.
- the **Monitoring Client** (collects monitoring information);
  - Collects infrastructure monitoring data (as specified in each GBDL) by using different *Monitoring Clients*.
  - Infrastructure monitoring data can be used to interpret benchmark results based on the state of the infrastructure during benchmark execution.

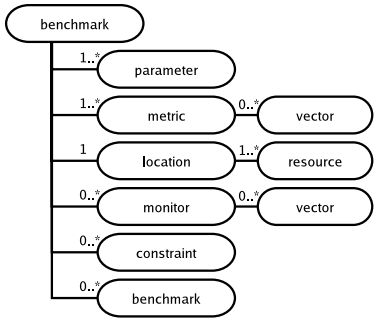
2.3 The GridBench Definition Language

The GridBench Definition Language was introduced to the system for several reasons:

- To allow easy definition of benchmarks, including work-flow benchmarks;
- To introduce a middleware-independent definition of benchmarks;
- To serve as a container for associating a definition to the resulting metrics as well as the collected monitoring data.

Figure 3 provides a high-level schematic view of the GridBench Definition Language. The benchmark definition includes all necessary information needed to run a benchmark. It includes a set of *parameters*, which specify details for the benchmark execution (such as the path to the executable and benchmark-specific parameters). It also contains a *location* which specifies the resources on which it should run. A *benchmark* can be hierarchical in nature, meaning that

[1] A *model* definition is a template benchmark definition with default parameters. A model is used for the creation of a new benchmark definition.




```
<benchmark name="epwhetstone"
          date="20040515023918"
          type="mpi" >
 <location>
   <resource name="cluster.ui.sav.sk"
         cpucount="16"
          wncount="16"
       jobmanager="jobmanager-pbs-workq"/>
 </location>
 <parameter name="executable" type="value"
        dataType="0">epwhetstone</parameter>
 <parameter name="execpath" type="value"
        dataType="0">/opt/cg/gb/bin</parameter>
 <parameter name="stage_executable" type="value"
        dataType="0">manual</parameter>
 <parameter name="nloops" type="value"
       dataType="1">10000</parameter>
</benchmark>
```

Fig. 4. Left: A schematic overview of GBDL; shown in boxes are the main parts of a GBDL document. Right: An example GBDL definition

it can be made up of other *benchmarks*. This, in conjuction with the use of the execution *constraint* elements, can be used to specify simple workflows. A benchmark *metric* may be in the form of a single value or in the form of a *vector* of values (such as bandwidth at different packet-sizes).

2.4 Archiver Web-Service

The *Archiver* allows the storage and retrieval of results generated by executions of the GridBench Suite Benchmarks through the Gridbench Framework.

The *Archiver* was introduced in order to serve the following purposes:

- To manage a potentially large number of results depending on the size of the Grid under study, the number of benchmarks and the frequency of their execution.
- To provide a central repository for the results allowing access to measurements for users or Grid services.
- To hold a set of *model* definitions serving as customizable benchmark definitions.

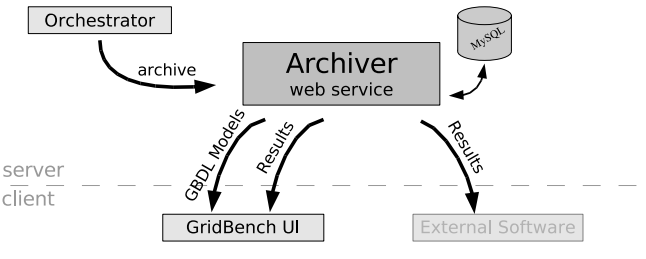


Fig. 5. Diagram describing the Archiver functionality

The *Archiver* is an *interface* implemented as a web service. The *Archiver* interface may have several implementations depending on the back-end in use. There are already implementations for using the *Apache Xindice* native XML database as a back-end and the (newer) *MySQLArchiver* implementation using the MySQL relational database as a back-end.

2.5 Orchestrator Web-Service

When a new benchmark description (in the form of GBDL) is delivered to the *Orchestrator* web service for execution the GBDL is translated to the Job Description Language required by the underlying middleware. All specified monitoring data collection is initiated and the job is submitted. When the job finishes, it's output (the metrics) are incorporated into the benchmark, as well as all the collected monitoring data. The final GBDL is then archived using the Archiver service.

The diagram in figure 6 describes the Orchestrator functionality in a series of steps. The steps are given below (the numbers correspond to the circled items in the diagram):

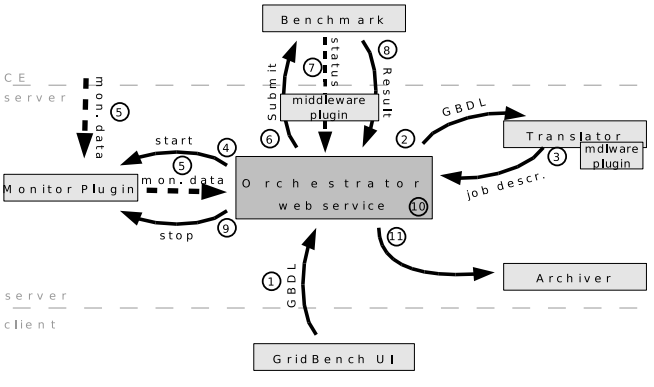


Fig. 6. Diagram describing the Orchestrator functionality

1. The *Orchestrator* receives a benchmark description in the GridBench Description Language (XML). This will originate from the GridBench GUI or from an automated system performing automated / periodic executions;
2. The GBDL is passed to the *GBDL translator* (which is part of the *Middleware Plugin*) which generates a middleware-specific job description in the syntax and format required by the underlying middleware;
3. The middleware-specific job description is then returned to the *Orchestrator*;
4. The *Orchestrator* determines all monitoring that need to be performed, which is specified by the *monitor* element(s) of the GBDL. Using the *type* and *query* attributes of the *monitor*, the correct monitoring plugin is invoked.

5. Monitoring data collection is started. (In the event where the banchmark is put in the target resource's local queue, synchronization of monitoring data collection and the actual benchmark execution is performed by job-status monitoring);
6. The benchmark job is then submitted using the *Middleware plugin*;
7. The benchmark job's status is monitored either by an "in-process wait" or by polling;
8. The benchmark job finishes and the result (i.e. the standard output containing the *metrics*) is returned to the *Orchestrator* by the *Middleware Plugin*;
9. The *Monitoring Plugin* is then signaled to stop collecting monitoring data and the collected data is returned to the *Orchestrator*;
10. The results of the benchmark in the form of *metric* elements, as well as the monitoring data, are incorporated into the original GBDL. If the *resources* specified in the *location* element were not specified explicitly (i.e. resources were allocated by the system) then location element is also updated;
11. Finally, the resulting GBDL is passed to the Archiver, concluding the *Orchestrator*'s role as it relates to this specific benchmark.

2.6 The GridBench User Interface: The "Virtual Workbench"

GridBench provides a user-friendly graphical interface for defining and executing benchmarks, as well as browsing results. Additionally it provides tools for result analysis through the easy construction of custom graphs from archives results. Figure 8 shows the main graphical use interface for the definition of benchmarks.

In Figure 8 we can observe the list of available benchmarks (the list on the left) and the available resources (the list on the right). The resource list shows resources retrieved from one or more Grid Information Systems (MDS), with details about each resource's composition such as free/busy CPU's and Worker nodes, dual/single CPU machines etc. Additionally a set of tests can be performed on each resource. In Figure 8 we can see tests such as the "PBS" test and the "MPI" tests. These tests will test each resource for correct configuration of PBS and MPI respectively. Tests involving multiple sites (e.g. using MPICH-G2) can also be performed. Such tests are usefull for detecting configuration problems as well as connectivity/firewall issues. More tests (e.g. targetting other local queuing systems) can be easily added by implementing simple Java interfaces.

Defining and executing a benchmark is as easy as dragging a benchmark onto one of the resources (shown in Figure 8). The user has the opportunity to tune the benchmark parameters prior to execution via a benchmark configuration panel. The user can easily construct graphs as the ones in the results section by using the "result matrix" shown in Figure 7.

3 Use-Case Scenarios

We present 2 simple use-case scenarios for GridBench in order to illustrate the functionality visible to the end-user and the overall simplicity in using the tool to get performance metrics for Grid resources. First we describe the scenario where a user would like to get a "picture" of the currest status of a set of resources in terms of low-level performance metrics. In the second case the user has a specific application in mind and would like to select a resource onto which to execute the application. Many other use-case scenarios are possible; in fact some do not even involve an end-user. For example, metrics obtained through GridBench mechanisms can be used by a scheduler that performs resource ranking on an application basis in a way that is completely transparent to the user.

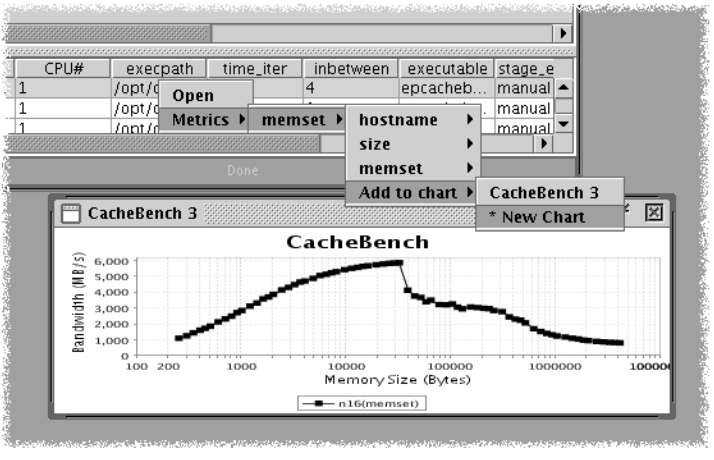


Fig. 7. The GridBench graphical user interface showing the generation of charts from historical data. The result shown is from a cache benchmark

3.1 Use-Case Scenario 1: Comparing Resources

As a first use-case, we consider a user who wants to compare a set of resources in terms of 2 "basic" performance factors : CPU FLOP/s and memory bandwidth. The user would like to use "fresh" data so she opts to invoke new benchmark executions instead of fetching historical data. The user can perform the following steps:

1. Determine which metrics will tell you what you want to know about the resources. In this case, the metrics for these factors can be delivered by a set of benchmarks as summarized below:

Factor	Metric	micro-benchmark
CPU	OP/s	EPWhetstone
Memory	bandwidth	EPStream

2. Using the GridBench GUI simply drag each of the benchmarks onto each resource and submit the benchmark (Figure 8). When the benchmark execution finishes, the result will be automatically archived.
3. Using the GridBench GUI put together comparative charts for the resources for each benchmark (Figure 9).

From the results on Figure 9 (the charts were generated using the GridBench GUI) we observe that the three sides that were chosen for comparison vary in

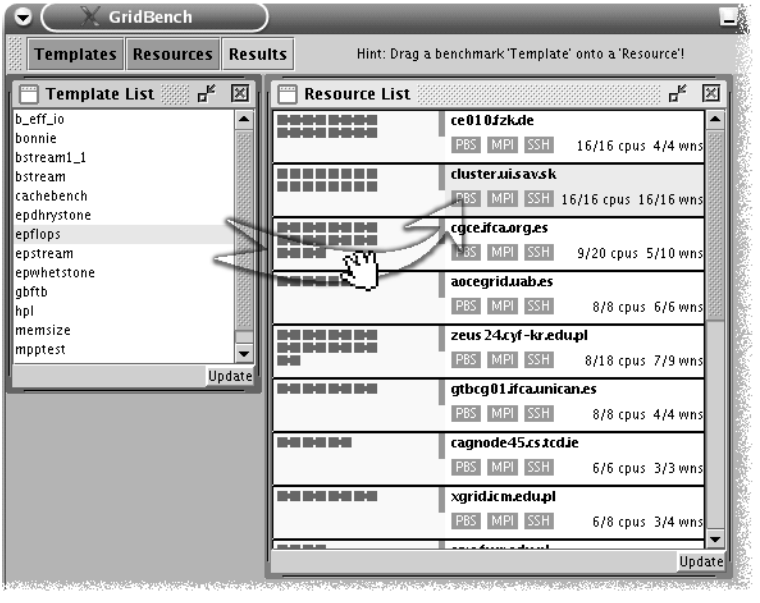
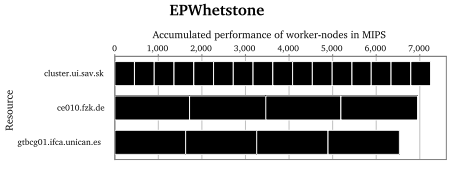
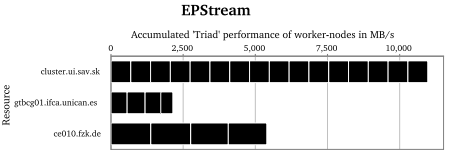


Fig. 8. Screen-shot of the GridBench graphical user interface. The list on the left is a list of benchmarks that are integrated into GridBench. The list on the right shows the currently available resources and their status in terms of busy/free CPU's. Invoking a benchmark on a resource is as simple as dragging a benchmark from the template list to a resource in the resource list



(a) Aggregate CPU performance



(b) Aggregate memory bandwidth

Fig. 9. Results for use-case scenario 1

their measurements. At this point it is important to note that two of the resources (`ce010.fzk.de` and `gtbcg01.ifca.unican.es`) use dual-CPU worker-nodes. In terms of aggregate CPU performance they vary only sightly. In terms of memory bandwidth the performance varies greatly as shown in figure 9(b) (probably due to the memory technology employed at each resource). Considering a memory-intensive application where the main requirement is memory bandwidth then a user (or resource broker) can select the four worker nodes from `ce010.fzk.de` rather than the four from `gtbcg01.ifca.unican.es`.

3.2 Use-Case Scenario 2: Application Performance

As a second use-case we consider a user that wants to compare resources based on performance of a given application or kernel [2]. The user, in this case a surgeon, needs to find the best resources to run a set of simulations. The user has a given application that is used *frequently*, it is therefore justifiable to perform some trivial instrumentation/timings on the application's computational kernel (e.g. to measure iteration times or simply measure completion time on a given dataset) and make it part of the benchmarks available in GridBench.

One of the primary design goals of the GridBench framework is the easy inclusion of new benchmarks/kernels. In this use-case scenario the user wishes to include a frequently used kernel; the following steps need to be taken:

1. Create a new GBDL description (model) and add it to the Archiver database;
2. Write a simple implementation of the ParameterHandler Java interface;
3. Instrument the code of the kernel to generate metrics.

A New GBDL Description

The first step in adding a new kernel is to create a new GBDL description such as the one that follows:

```
<benchmark name="bstream1_1" date="" type="mpi"
           model="true" description="B_stream 1.1 ..." >
  <parameter name="executable" type="system">bstream1.1</parameter>
  <parameter name="iterations" type="value">40</parameter>
  <parameter name="Reynolds" type="value">20</parameter>
  <parameter name="data_id" type="value">tube38x40x40</parameter>
  <parameter name="stage_file" type="system">tube38x40x40.bs</parameter>
 </benchmark>
```

This description states that:

- this is a benchmark description that is to be used as a model (*model="true"*);
- the parameters *iterations*, *Reynolds* and *data_id* are application-specific parameters required by the kernel executable;
- "bstream1.1" is the name of the *executable* and file "tube38x40x40.bs" needs to be staged;

[2] The kernel in question is from a medical application, developed at the University of Amsterdam, for pre-operative planning of vascular reconstruction. It involves blood-flow simulation using a Lattice Boltzmann method in arteries using 3-Dimensional data obtained from MRI scans of the patient [9].

Since the formatting of command-line arguments to the application executable is application-dependent the user needs to provide a ParameterHandler.

Writing a ParameterHandler

A benchmark-specific ParameterHandler is required for special formatting of command-line arguments (or creation of parameter files etc). In this use-case scenario the applications takes three parameters, which need to be provided in a given order on the command-line. A typical invocation would be:

```
bstream1_1 20 tube38x40x40 40
```

During translation of the GBDL to the middleware-specific job description, the class *ParameterHandler_**bstream1_1*** will be dynamically loaded:

```
public class ParameterHandler_bstream1_1 implements ParameterHandler{
  public java.util.Vector getCommandLineArguments(Benchmark benchmark){
    ParameterCollection parameters=benchmark.getParameters();
    Vector parameterVector=new Vector();

    Parameter data_id=parameters.getParameter("data_id");
    Parameter reynolds=parameters.getParameter("Reynolds");
    Parameter iterations=parameters.getParameter("iterations");

    parameterVector.add(raynolds.getValue());
    parameterVector.add(data_id.getValue());
    parameterVector.add(iterations.getValue());

    return parameterVector;
  }
  ...
}
```

The method *getCommandLineArguments()* is called and returns an ordered list of parameters correctly formatted and ready to be passed to the application executable.

Instrumenting Application Codes

Instrumentation of codes is highly application-specific and usually involves trivial modification of the source code to obtain timings at a high level. In our specific use-case the application performs iterations which are controlled by a main loop. In total, about ten lines of code were added in order to time each iteration and output the following metrics onto the standard output:

```
<metric  name="iteration_times" type="vector" unit="s" step="20" period="200">
  <vector name="time">0.079617 0.079529 0.079511 0.079498 ... 0.094326</vector>
</metric>
<metric  name="completion_time" type="value" unit="s">639.633215</metric>
```

Obtaining Measurements

Once the kernel has been integrated into GridBench the user can invoke it just like any other benchmark. The same steps listed in the previous use-case apply to this case as well. One difference is that now the kernel benchmark takes considerably longer to run (tens of minutes) than the micro-benchmarks (a few seconds) in the previous use-case. In this case the user opts to use previously archived executions of the kernel benchmark because it is considerably expensive to get fresh measurements. The steps are now:

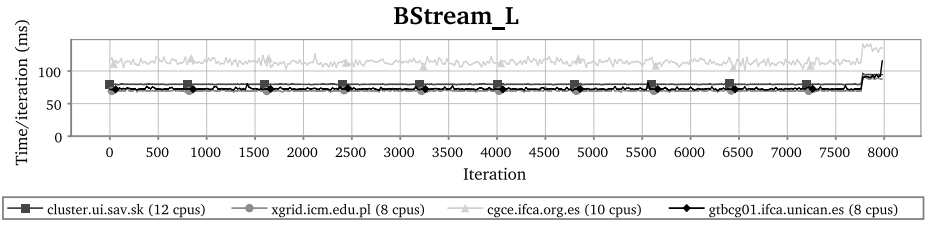


Fig. 10. Results for use-case 2, showing iteration times of a given kernel on four resources

1. Retrieve archived results for this kernel;
2. Benchmark the resources for which there are no archived results;
3. Compare the results.

Invoking the kernel benchmark on a set of resources allows us to construct the chart shown in Figure 10. Based on these results a user (or resource broker) can make relatively safe decisions for resource selection, given that the criterion for a "good" selection is the performance of the given kernel.

4 Related Work

The ALU-Intensive Grid Benchmarks [3] are a specification to run the the NAS Parallel Benchmarks [1] in pre-defined workflows and from that infer the performance of Grid systems. The AIGB aim to benchmark the ability of dynamic collections of Grid resources to executes several types of workflow, while the GridBench suite proposed an more hierarchical approach both in terms of infrastructure and of benchmark types. Nevertheless, the GridBench tool could serve as a means to execute these benchmarks just like any other benchmark.

Diperf [4] is a distributed performance-testing framework aimed at automating performance evaluation of services. It does not address computational resources or network performance directly.

Also, work has been done to "assess" the Grid using "probes" [2] but this work focuses mainly on file transfers, remote execution, and Information Service responses. Computational resource performance is not addressed.

5 Conclusions and Future Work

We have provided an overview the GridBench services and user interface which can serve as a "virtual workbench" for performing benchmarking experiments, archiving benchmark specifications and results and an aid for analysis of metrics.

We have presented two elementary use-case scenarios and illustrated the ease of use of the tool: The first use-case illustrated how end-users and administrators can perform benchmarking experiments either for resource selection of for determining the operational status of resources. The second use-case illustrated how a user or application developer can obtain results from new application-based benchmarks using the GridBench framework.

In on-going and future work we are working on the implementation of more benchmarks focusing on the aspects of availability and performability and the derivation of higher-level metrics to express "quality features" of Grid infrastructures: Homogeneity, trustworthiness of GIS, health of the infrastructure, reliability and robustness. We also plan to enrich the Gridbench suite with more benchmarks based on existing Grid applications.

We plan to extend the GBDL specification to include constrained and automatic parameter selection and to include additional middleware plugins to provide interoperability with more infrastructures (such as UNICORE).

Acknowledgments

This work was supported by the European Union through the CrossGrid project (IST-2001-32243). The authors wish to acknowledge Alfredo Tirado-Ramos and Lilit Abrahamyan (University of Amsterdam) for the blood flow application code, and the support of the CrossGrid testbed team for running the distributed simulation.

References

1. David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, and Maurice Yarrow. The nas parallel benchmarks 2.0. *The International Journal of Supercomputer Applications*, 1995.
2. Greg Chun, Holly Dail, Henri Casanova, and Allan Snavely. Benchmark probes for grid assessment. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA*. IEEE Computer Society, 2004.
3. R.F Van der Wijngaart and Michael Frumkin. Alu intensive grid benchmarks. https://forge.gridforum.org/projects/gb-rgs, 2004.
4. Catalin Dumitrescu, Ioan Raicu, Matei Ripeanu, and Ian Foster. Diperf: an automated distributed performance testing framework. In *Proceedings of the 5th International Workshop on Grid Computing (GRID2004)*. IEEE, November 2004.
5. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symp. on High-Performance Distributed Computing*, pages 365–375. IEEE Computer Society, 1997.
6. The EU CrossGrid Project. http://www.eu-crossgrid.org.
7. The EU DataGrid Project. http://www.eu-datagrid.org.
8. The LCG Project. http://lcg.web.cern.ch/LCG/.

9. P.M.A. Sloot, A. Tirado-Ramos, A.G. Hoekstra, and M. Bubak. An interactive grid environment for non-invasive vascular reconstruction. In *2nd International Workshop on Biomedical Computations on the Grid (BioGrid'04), in conjunction with Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)*, Chicago, Illinois, USA, April 2004. IEEE. CD-ROM IEEE Catalog # 04EX836C.
10. George Tsouloupas and Marios D. Dikaiakos. Gridbench: A tool for benchmarking grids. In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, pages 60–67, Phoenix, AZ, November 2003. IEEE.
11. George Tsouloupas and Marios D. Dikaiakos. Characterization of computational grid resources using low-level benchmarks. Technical Report TR-2004-5, Dept. of Computer Science, University of Cyprus, 2004.

A Method for Estimating the Execution Time of a Parallel Task on a Grid Node

Panu Phinjaroenphan[1], Savitri Bevinakoppa[1], and Panlop Zeephongsekul[2]

[1] School of Computer Science and Information Technology,
[2] School of Mathematical and Geospatial Sciences,
RMIT University, GPO Box 2476V, Melbourne Australia
{pphinjar, savitri}@cs.rmit.edu.au
panlopz@rmit.edu.au

Abstract. The mapping problem has been studied extensively and many algorithms have been proposed. However, unrealistic assumptions have made the practicality of those algorithms doubtful. One of these assumptions is the ability to precisely calculate the execution time of a task to be mapped on a node before the actual execution. Since the theoretical calculation of task execution time is impossible in real environments, an estimation methodology is needed. In this paper, a practical method to estimate the execution time of a parallel task to be mapped on a grid node is proposed. It is not necessary to know the internal design and algorithm of the application in order to apply this method. The estimation is based upon past observations of the task executions. The estimating technique is a k-nearest-neighbours algorithm (knn). A backward predictor elimination, leave-one-out cross validation, and a statistical technique are used to derive the relevant parameters to be used by knn. Experimental results show that on average the proposed method can produce 2.3 times the number of accurate estimated execution times (with errors less than 25%) greater than the existing method.

1 Introduction

Computational grid has been introduced as a new distributed computing paradigm that is able to interconnect heterogeneous networks and a large number of computing nodes regardless of their geographical locations [1]. This new paradigm provides an access to tremendous computational power that can be harnessed for various applications. Parallel applications are developed to solve implementations of computational intensive engineering or scientific problems that require such power.

The main aim of solving such problems with a parallel application is to reduce the execution time. As a computational grid involves a large number of nodes, one of the challenging problems is to decide the destination nodes where the tasks of the application are to be executed. This process is formally known as the *mapping problem* [2]. In this paper, we broadly categorise studies of the mapping problem into two classes: *practical* and *theoretical*.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 226–236, 2005.

In practical studies, the focus is on investigating an efficient approach to map a *specific* parallel application on real environments. GrADS [3] and CGRS [4] are examples of practical mapping studies. The internal knowledge of the application, such as design and algorithm need to be known.

In theoretical studies, the problem is usually modelled at an abstract level using graphs, and internal knowledge of the application is assumed to be unknown. The developed mapping approach is therefore *generic*, but cannot be used in reality due to unrealistic assumptions of the graph based model.

The current situation indicates a lack of a *practical* and *generic* mapping approach. A methodology that can be undertaken to develop one such approach is to address the unrealistic assumptions in theoretical studies. One of these is the ability to precisely calculate the execution time of a task to be mapped on a node before the actual execution. In practice, such a calculation is impossible; however, an estimate (i.e. *estimated execution time*) can be made. This problem is formally called the *execution time estimation problem* [6].

In this paper, a method is developed to estimate the execution time of a parallel task, a task with inter-task communication, on a grid node. We only assume that the input problem size (e.g. the sizes of the matrices in a matrix-matrix multiplication application), the number of tasks, and the topology of the application are known. The estimation method is based on past observations of the task executions. A k-nearest-neighbours (*knn*) algorithm is employed as the estimating technique. The relevant parameters to be used by *knn* are dynamically and automatically chosen using the combination of a *backward predictor elimination*, a *statistical technique* and *leave-one-out cross validation* [5].

In the experiments, the proposed method is compared with the existing estimation method presented in [6] by estimating the execution times of the tasks of a matrix-matrix multiplication application developed with Cannon's algorithm (Cartesian topology). Experimental results show that on average the proposed method can produce 2.3 times the number of accurate estimated execution times (with error less than 25%) greater than the existing method.

2 Related Work

The solutions to the execution time estimation problem are categorised into *code analysis*, *analytic benchmarking and code profiling* and *past observations* [6].

The first two classes assume that the internal design and algorithm of the application are known. The user-supplied performance model used in GrADS [3] and CGRS [4] fit into these categories. On the other hand, estimation based upon past observations does not require any knowledge of the internal design and algorithm. However, some previous observations are essential.

An estimation method based upon past observations is proposed in [6]. The employed estimating technique is a k-nearest-neighbours algorithm. Even though their experimental results suggest a promising method, there are some shortcomings. Their method is inflexible since the number of predictors (variables used to make an estimate) is fixed. Another restriction assumption is that the exe-

cution time of a task only depends on the performance of the node the task is mapped on, and the input problem size. However, in practice, the execution time of a task may depend on its communications to other tasks. Another limitation of this method is that the number of neighbours (k) chosen is always equal to $n^{\frac{4}{5}}$, where n is the number of known observations, and no justification has been offered as to the choice of this number.

3 Estimation Based upon Past Observations

The execution time of a task on a given node depends on a vector $\mathbf{x}$ of p predictors, $\mathbf{x}^\top = (x_1\ x_2\ ...\ x_p)$. Given y as the execution time of a task, y is considered to be a function of $\mathbf{x}$.

$$y = f(\mathbf{x}^\top) \tag{1}$$

Some vectors of predictors and their corresponding execution times are known. Let $\mathbf{X}$ and $\mathbf{y}$ represent n of these vectors, respectively.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1p} \\ x_{21} & x_{22} & ... & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & ... & x_{np} \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{2}$$

The goal is to estimate the execution time $\hat{y}$ (*dependant*) from a vector of predictors $\mathbf{x}_q$ (*query point*) using the known observations $\mathbf{X}$ and $\mathbf{y}$ (*dataset*), and the percentage relative residual error ($\%e$) is used to evaluate the accuracy, i.e.

$$\%e = \frac{|\hat{y} - y|}{y} \cdot 100. \tag{3}$$

4 The Proposed Estimation Method

The proposed method consists of two processes: *estimating* and *learning*. The former is to estimate the execution time of the query point. The latter is to derive the relevant parameters to be used in the estimating process.

4.1 Estimating Process

Given $\mathbf{x}_q$ as the query point, with knn, $\hat{y}$ is the average of the other k execution times which are nearest neighbours of $\mathbf{x}_q$, i.e.

$$\hat{y} = \frac{\sum_{j=1}^{k} y_j}{k}. \tag{4}$$

The other k execution times are determined from the Euclidean distance, $d(\cdot)$, of their predictors to the query point $\mathbf{x}_q$, which is given by

$$d(\mathbf{x}, \mathbf{x}_q) = \sqrt{\sum_{i=1}^{p} (w_i \cdot (x_i - x_{q_i}))^2} \tag{5}$$

where x_i and x_{q_i} are the i^{th} predictor in $\mathbf{x}$ and $\mathbf{x}_q$, respectively. A *distance factor* w_i, is used to multiply the i^{th} predictor to represent how important a predictor is (the greater the distance factor, the more the important).

Using distance as a criterion, it is better to give greater weight to observations that are close to $\mathbf{x}_q$ and less weight to those that are remote. To assign the weight to an observation, a weighting (kernel) function is necessary. The Gaussian kernel is used as the weighting function and is given by

$$K(d) = e^{-d^2}. \tag{6}$$

Using this, $\hat{y}$ is now the weighted average of the execution times of k nearest neighbours and is given by

$$\hat{y} = \frac{\sum_{j=1}^{k} y_j K(d(\mathbf{x}_j, \mathbf{x}_q))}{\sum_{j=1}^{k} K(d(\mathbf{x}_j, \mathbf{x}_q))}. \tag{7}$$

4.2 Learning Process

It can be seen that the predictors, the number of neighbours, and the distance factors need to be defined for the *knn* in the estimating process. The learning process explained here is for specifying these parameters. The process consists of two steps: *preprocessing* and *parameter-deriving*.

Preprocessing: Let $\mathbf{p}$ represent a predictor type, and $\mathbf{p}^\top = (x_1\ x_2\ ...\ x_n)$. Hence, $\mathbf{X}$ in (2) can now be rewritten as

$$\mathbf{X} = \left[\, \mathbf{p}_1\ \mathbf{p}_2\ ...\ \mathbf{p}_p \,\right] = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1p} \\ x_{21} & x_{22} & ... & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & ... & x_{np} \end{bmatrix}. \tag{8}$$

The first step is to transform the elements in each $\mathbf{p}$ such that their values range from zero to one using (9).

$$x_i = \frac{x_i - min_x}{max_x - min_x} \tag{9}$$

where x_i, max_x, and min_x are the i^{th}, the maximum, and the minimum elements in $\mathbf{p}$, respectively.

The second step is to remove the predictors that have no influence on the dependants $\mathbf{y}$. $\mathbf{p}$ has no influence on $\mathbf{y}$ if all elements in $\mathbf{p}$ are identical. This situation usually occurs when the number of observations in the dataset is small.

The final step is to remove *multicollinearity*. Multicollinearity refers to the situation that a pair of predictors are highly correlated, in which one of them can be ignored. The linear relationship between predictors $\mathbf{p}_i$ and $\mathbf{p}_j$ can be measured from their *correlation coefficient* (r_{ij}) [7], which is given by

$$r_{ij} = \frac{n \sum x_i x_j - \sum x_i \sum x_j}{\sqrt{[n \sum x_i^2 - (\sum x_i)^2][n \sum x_j^2 - (\sum x_j)^2]}}. \tag{10}$$

00. min_{cv}, k_{opt}, l_{opt} = **LOOCV** (**X**, **y**)
01. **for** ($l = 1$; $l \leq max_l$; $l = l + 1$)
02. calculate w for each **p**;
03. **for** ($i = 1$; $i \leq n$; $i = i + 1$)
04. $\mathbf{x}_q = \mathbf{x}_i$;
05. $y = \mathbf{y}_i$;
06. $\mathbf{X} = \mathbf{X} - \mathbf{x}_i$;
07. $\mathbf{y} = \mathbf{y} - \mathbf{y}_i$;
08. **for** ($j = 1$; $j \leq n - 1$; $j = j + 1$)
09. $D[j][l] = d(\mathbf{x_j}, \mathbf{x_q})$;
10. sort D by distance;
11. **for** ($k = 1$; $k \leq max_k$; $k = k + 1$)
12. $\hat{y} = knn$ from k neighbours based on the distances in D;
13. $E[i][k][l] = \frac{|\hat{y}-y|}{y} \cdot 100$;
14. $\mathbf{X} = \mathbf{X} + \mathbf{x}_q$;
15. $\mathbf{y} = \mathbf{y} + y$;
16. min_{cv} = minimum cv in E;
17. k_{opt}, l_{opt} = k^{th} and l^{th} indices that minimum cv is found;

Fig. 1. Leave-one-out cross validation algorithm

In our study, if $|r_{ij}|$ is greater than 0.90 then a multicollinearity exists between them.

Parameter-deriving: In this step, the predictors are related to the dependants in the dataset as to derive the actual predictors ($\mathbf{X}_{act}$ – the predictors that will be used in the estimating process), and the optimal number of neighbours (k_{opt}), and distance factors ($\mathbf{w}_{opt}$). The main technique to accomplish this is *leave-one-out cross validation (LOOCV)* [5]. The algorithm is shown in Fig.1, which is to leave the i^{th} observation out of the dataset, and use the other observations to estimate the left out observation. The objective is to find **X**, k, and **w** that minimise the cross validation (cv) function

$$cv(\mathbf{X}, k, \mathbf{w}) = \frac{\sum_{i=1}^{n} \%e_i}{n} \tag{11}$$

where $\%e_i$ is the percentage relative residual error of the i^{th} observation when it is left out and estimated by using **X**, k, and **w**.

The distance factor for each predictor in the algorithm is derived from the statistical technique *Spearman's rank correlation coefficient* (r_s) [7], and the *level of importance*.

r_s is used to measure the strength of association between two sets of data, assuming that the underlying relationship is unknown. r_s is given by

$$r_s = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n^3 - n} \tag{12}$$

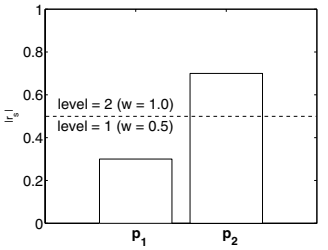


Fig. 2. $|r_s|_1$ and $|r_s|_2$ are 0.3 and 0.7, respectively. Since $l = 2$, $m_1 = 1$ and $m_2 = 2$, and w_1 and w_2 are 0.5 and 1.0, respectively

where d is the difference in *statistical rank* – the ordinal number of a value in a list arranged in increasing order – of corresponding variables. The values of r_s range from -1 to 1, indicating perfect negative and positive association, respectively. Let $|r_s|_i$ represent $|r_s|$ between the i^{th} predictor and the dependants $\mathbf{y}$, this predictor is in level m if and only if

$$\frac{m-1}{l} < |r_s|_i \leq \frac{m}{l} \tag{13}$$

where l is the number of levels and $m = 1, ..., l$. Given m_i as the level that the i^{th} predictor is in, the distance factor for this predictor, w_i, is given by

$$w_i = \frac{m_i}{l}. \tag{14}$$

Fig.2 shows an example of how to calculate the distance factors. In $LOOCV$, max_l is the predefined maximum number of levels while max_k is the predefined maximum number of neighbours. The possible numbers of neighbours range from 1 to $n-1$.

The execution time of each left out observation is estimated over different ks and ls, and the associated error ($\%e$) is stored in an array E. After all the errors have been stored, the algorithm calculates cv (over different ks and ls), and returns the minimum cv and the k^{th} and l^{th} indices (as k_{opt} and l_{opt}) that lead $LOOCV$ to min_{cv}.

Thus far, the predictors given to $LOOCV$ are the ones after the preprocessing step. However, they are not yet the actual predictors. To derive the actual predictors, the simplest approach is to generate all possible combinations of the predictors, process each to $LOOCV$, and pick the one that yields the least cv.

However, a problem arises when p is large since the complexity grows exponentially with the number of predictors, i.e. $O(2^p)$. To address this problem, the *backward predictor elimination* is augmented into the *parameter-deriving algorithm* (as shown in Fig.3) to seek the actual predictors.

The idea is to drop each predictor one by one to make p new sets of predictors. If all cvs from processing these sets to $LOOCV$ are more than the cv from processing all predictors in $\mathbf{X}$ to $LOOCV$, the predictors in $\mathbf{X}$ are the actual

00. $\mathbf{X}_{act}$, k_{opt}, l_{opt} = **parameter-deriving** ($\mathbf{X}$, $\mathbf{y}$)
01. min_{cv}, k_{opt}, l_{opt} = LOOCV($\mathbf{X}, \mathbf{y}$);
02. $\mathbf{X}_{act} = \mathbf{X}$;
03. **for** ($i = 1$; $i \leq p$; $i = i + 1$)
04. tmp_{cv}, tmp_k, tmp_l = LOOCV($\mathbf{X} - \mathbf{p}_i, \mathbf{y}$);
05. **if** ($min_{cv} \leq tmp_{cv}$)
06. $min_{cv} = tmp_{cv}$;
07. $\mathbf{X}_{act} = \mathbf{X} - \mathbf{p}_i$;
08. $k_{opt} = tmp_k$;
09. $l_{opt} = tmp_l$;
10. **if** ($\mathbf{X}$ is $\mathbf{X}_{act}$)
11. return;
12. **else**
13. $\mathbf{X}_{act}$, k_{opt}, l_{opt} = parameter-deriving($\mathbf{X}_{act}, \mathbf{y}$);

Fig. 3. The parameter-deriving algorithm

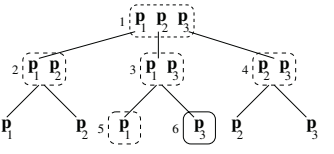


Fig. 4. Only the sets of predictors in the oval are considered in the backward predictor elimination. Here, $\mathbf{X}_{act} = \mathbf{p}_3$ and $cv(\mathbf{X} = [\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3]) \geq cv(\mathbf{X} = [\mathbf{p}_1\mathbf{p}_3]) \geq cv(\mathbf{X} = [\mathbf{p}_3])$

predictors. Otherwise, the *parameter-deriving* is recursively called with the set that yields the least cv (see Fig.4 for an example). The returned $\mathbf{X}_{act}$, k_{opt}, and l_{opt} are the actual predictors, the optimal number of neighbours, and levels which is used to derive the optimal distance factors, respectively. The complexity of the *backward predictor elimination* is $O(\frac{p \cdot (p+1)}{2})$ for the worst case.

5 Experiments

In the experiments, three estimation methods, as shown in Table 1, are evaluated. Method-1 is the estimation method proposed in [6]. In method-3, max_k is set to $n - 1$, which is the maximum possible number of neighbours, and max_l is set to 10. max_k in method-2 is also set to $n-1$. Notice that method-2 is a specialisation of method-3. The simulator used in the experiments is GMap[1]. All the experiments are conducted on a 2.8 GHz Intel Pentium-4 computer.

[1] GMap is a simulator developed to study the mapping problem, available at http://www.cs.rmit.edu.au/~pphinjar/GMap

Table 1. The experimented estimation methods

method	learning process	estimating process
method-1	fixed $\mathbf{X}$, $k_{opt} = n^{\frac{4}{5}}$, and $l_{opt} = 1$	*knn* explained in [6]
method-2	dynamically chosen $\mathbf{X}$ and k_{opt}, and $l_{opt} = 1$	*knn* explained in Sect.4.1
method-3	dynamically chosen $\mathbf{X}$, k_{opt}, and l_{opt}	*knn* explained in Sect.4.1

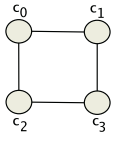


Fig. 5. The topology of the experimented parallel application. Here, c_1 and c_2 are the neighbour tasks of c_0, c_0 and c_3 are the neighbour tasks of c_1, and so on

5.1 Parallel Application

The execution times of the tasks of a four-task square matrix-matrix multiplication are estimated. The topology of the application is Cartesian (as shown in Fig.5). The algorithm is Cannon's algorithm (see [8] for details), in which the tasks perform some different instructions.

As for simplicity, the small number of tasks is experimented on. However, the same method can be applied directly to applications with larger number of tasks and other types of topologies.

5.2 Grid Testbed

The experimental testbed is partially modelled from ThaiGrid testbed [9]. The testbed consists of 7 clusters 96 nodes and 141 processors. It is assumed that the clusters are located in 7 different countries, and all nodes are dedicated. Mapping tasks to unreliable nodes are not considered in the experiments.

The bandwidth among the nodes in the testbed are derived from the network statistics among nodes located in those 7 countries measured during May 2004, available from the PingER project [10].

5.3 Experimental Results

The application is executed on the testbed 100 times by randomly varying the size of the matrices (i.e. 100, 200, ..., 5000). At each run, 10 nodes are first randomly chosen. Then, the nodes to run the tasks are randomly chosen from these 10 nodes. This leads to multiple tasks being executed on the same node.

Assume that task c_0 (in Fig.5) is to be mapped on node v_0 while c_1 and c_2 (which are the neighbour tasks of task c_0) are mapped on nodes v_1 and v_2, respectively. The predictors used to estimate the execution time of task c_0 are the problem size of the application, the performance and *load factor* of nodes v_0, v_1 and v_2, and the bandwidth from node v_0 to nodes v_1 and v_2 and vice versa.

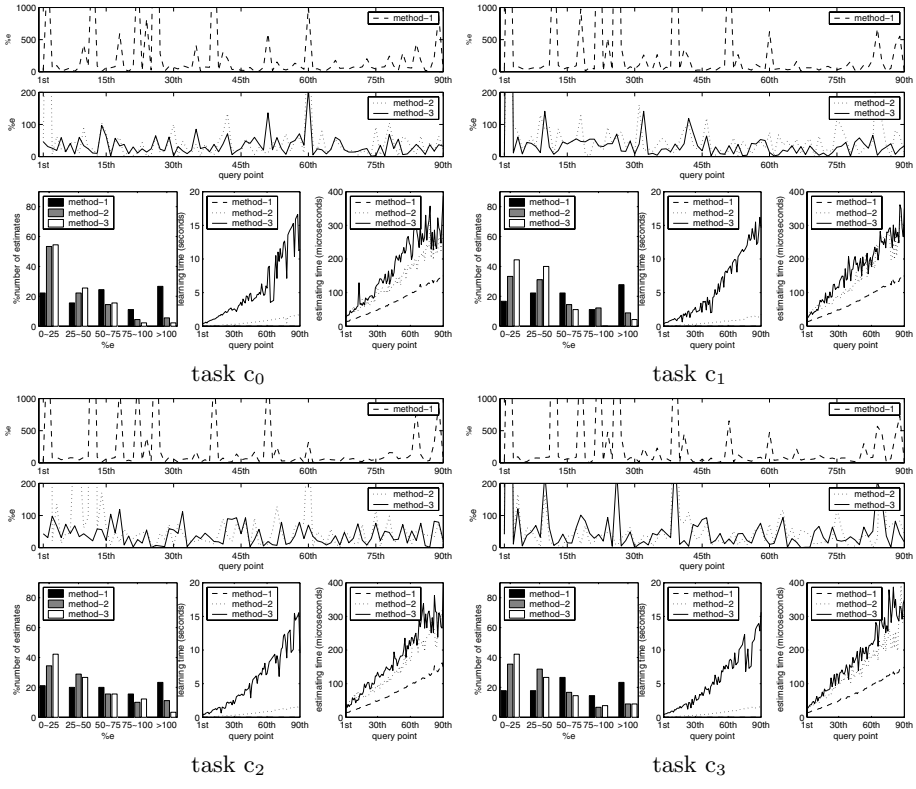


Fig. 6. Estimation evaluation for the application

The load factor of a node is defined as the upper bound of the ratio between the number of tasks to be mapped on that node and the number of processors of the node. For instance, if node v_0 has two processors, its load factor is $\lceil \frac{1}{2} \rceil = 1$.

The estimation starts from 10 observations in the dataset, and the total of 90 query points are estimated. Fig.6 shows the results from estimating the execution time of each task of the application.

Results from the top two sub-figures of each task show that $\%e$ of estimates produced from method-1 are very large when compared to method-2 and method-3. The more the number of observations in the dataset, the lower the $\%e$ of estimates produced from method-2 and method-3.

Results from the bar graphs of each task show that, in terms of estimation accuracy, method-3 outperforms the others. For example, from the results of task c_1, about 43%, 35%, and 18% of the total number of estimates have their $\%e$ less than 25%, which are considered as accurate estimates, when estimated with method-3, method-2, and method-1, respectively. Method-2 performs reasonably well whereas method-1 is the worst. On average, 27% of the total number of estimates produced from method-1 have their $\%e$ greater than 100% whereas around 10% and 6% with method-2 and method-3, respectively.

As expected, the learning time of method-3 is the greatest ($\simeq$ 15 seconds with 99 observations in the dataset) since it needs to perform cross validation to find $\mathbf{X}_{act}$, k_{opt}, and l_{opt}. Method-1 has the fastest learning as only adding the new observation to the dataset needs to be done. However, note that the learning can be done off-line. From the bottom right sub-figure of each task, results show that the estimating times of all methods are performed quite efficiently in the scale of less than 400 microseconds with 99 observations in the dataset.

It can be seen from the results that on average method-3 produces 2.3 times the number of accurate estimates ($\%e < 25\%$) greater than method-1.

6 Conclusions

In this paper, a new method to estimate the execution time of a parallel task on a grid node is proposed. This is to address the problem caused by an unrealistic assumption that the execution time of a task to be mapped on a node can be precisely calculated before the actual execution. The proposed estimation method is based upon past observations of the task executions. The employed estimating technique is a k-nearest-neighbours algorithm (knn). Leave-one-out cross validation technique, a backward predictor elimination, and a statistical technique are used to derive the relevant parameters to be used by knn. In the experiments, the proposed method is compared with the existing method by estimating the execution times of the tasks of a matrix-matrix multiplication developed with Cannon's algorithm (Cartesian topology). Experimental results show that on average the proposed method can produce 2.3 times the number of accurate estimates (with error less than 25%) greater than the existing method.

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for Future Computing Infrastructure. Morgan Kaufmann (1998)
2. Bokhari, S.: On the mapping problem. IEEE Transaction on Computers **C-30** (1981) 207–214
3. Dail, H., Berman, F., Casanova, H.: A decoupled scheduling approach for grid application development environments. Parallel and Distributed Computing **63** (2003) 505–524
4. Zhang, W., Fang, B., He, H., Zhang, H., Hu, M.: Multisite resource selection and scheduling algorithm on computational grid. In: 18^{th} International Parallel and Distributed Processing Symposium (IPDPS). (2004) 105–115
5. Atkeson, C., Schaal, S., Moore, A.: Locally weighted learning. AI Reviews **11** (1997) 11–73
6. Iverson, M., Ozguner, F., Potter, L.: Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. IEEE Transaction on Computers **48** (1999) 35–44

7. Walpole, R.: Introduction to Statistics. 3^{rd} edn. Collier Macmillan (1982)
8. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing. 2^{nd} edn. Pearson Education Limited, Essex, England (2003)
9. Varavithya, V., Uthayopas, P.: ThaiGrid: Architecture and overview. NECTEC Technical Journal **2** (2000)
10. The PingER Project, http://www-iepm.slac.stanford.edu/pinger/ (2004)

Performance of a Parallel Astrophysical N-Body Solver on Pan-European Computational Grids

Alfredo Tirado-Ramos[1], Alessia Gualandris[1,2], and Simon Portegies Zwart[1,2]

[1] Faculty of Science, Section Computational Science,
University of Amsterdam Kruislaan,
403, 1098 SJ Amsterdam, The Netherlands
[2] Astronomical Institute Anton Pannekoek,
University of Amsterdam Kruislaan,
403, 1098 SJ Amsterdam, The Netherlands

Abstract. We present performance results obtained by running a direct gravitational N-body code for astrophysical simulations across the Dutch DAS-2 and the pan-European CrossGrid computational grids. We find that the performance on large grids improves as the size of the N-body system increases because the computation to communication ratio becomes higher and a better load balance can be achieved. Communication among nodes residing in different locations across Europe becomes more evident as the number of locations increases. Nevertheless, contrary to our expectations, we find that the performance decreases only by about a factor three for a large simulation. We conclude that highly distributed computational Grid infrastructures can be used efficiently for simulating large gravitational N-body systems.

1 Introduction

Direct summation methods to model the dynamics and evolution of collisional systems allow scientists to follow the global evolution of large stellar systems along their lifetime. As a counterpart to their high numerical accuracy, they present $O(N^2)$ complexity, which translates in massive computational requirements for complete sets of inter-particle forces.

Significant improvement in the performance of direct codes used in the numerical simulation of astrophysical stellar systems can be obtained by means of general purpose parallel computers (Dorband, Hemsendorf and Merrit, [1]; Gualandris, Portegies Zwart and Tirado-Ramos, [4]), but the use of highly distributed clusters within computational grids has not yet been explored. Grid technology is rapidly becoming a major component of computational science. It offers a unified means of access to different and distant computational resources, with the possibility to securely access highly distributed resources that scientists do not necessarily own or have an account on. Connectivity between distant locations and interoperability between different kinds of systems and resources are some of the most promising characteristics of the Grid.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 237–244, 2005.

In this paper we present the results of the first experiments conducted on computational Grids using a direct gravitational N-body code. The code is parallelized with MPI using a systolic algorithm. We explore the effects of network latency on the performance on the DAS-2 Grid testbed [1], distributed within the Netherlands and running the Globus toolkit (Foster & Kesselman, [2]), as well as on the 18-node CrossGrid testbed [2], distributed across Europe (see Fig. 1) and running the LCG2 [3] infrastructure software.

In Sec. 2 we discuss the numerical integrator and the parallel algorithm used for our experiments. Section 3 briefly describes our experimental grid testbed setup, Sec. 4 presents the obtained timing results, and Sec. 5 contains our summary and conclusions.

2 Numerical Method

For the work described in this paper we consider a direct N-body code applied to the study of astrophysical stellar systems. An N-body code solves the equations of motion of N point particles interacting gravitationally with each other. In a direct method the gravitational force acting on each particle is computed by summing up the contributions from all the other particles according to Newton's law:

$$\mathbf{F}_i = m_i \mathbf{a}_i = -G m_i \sum_{j=1, j\neq i}^{j=N} \frac{m_j(\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^3}. \quad (1)$$

Starting from initial values of the positions and velocities of all the stars and using the values of the forces and their first derivatives, new positions and velocities at successive times can be computed. The code uses a fourth-order Hermite integrator (Makino & Aarseth, [9]) for the determination of the trajectories. This method results in an accurate integration of the equations of motion of all the stars and allows us to study the dynamical evolution of different stellar systems in great detail. On the other hand, the calculation has a $O(N^2)$ computational complexity and is therefore very demanding.

An important feature in our code is the use of the *hierarchical* or *block time-step* scheme (Makino [8]). The value of the step is computed for every particle after each force calculation, depending on the time-scale on which its orbital parameters change, and is quantized to a power of two. In this way, groups of particles are forced to share the same time-step and can be advanced at the same time. The particles sharing the same time-step are said to form a *block*.

We implemented the *ring* or *systolic algorithm* for a Hermite scheme with block time-steps using the standard MPI library package. The particles are evenly distributed among the processors during the initialization phase and the ones which need to be updated circulate among the nodes according to a virtual

[1] `http://www.cs.vu.nl/das2`
[2] `http://www.eu-Crossgrid.org`
[3] `http://lcg.web.cern.ch/LCG/Documents/default.htm`

ring topology. Partial forces are computed by the different nodes at each step and are summed up to obtain the total forces. The number of communication steps needed to compute the total forces is equal to the number of available processors. This algorithm has the advantage of minimizing memory requirements on each node.

3 Experimental Grid Setup

3.1 The DAS-2 Testbed

For our first set of experiments, we have used the Distributed ASCI Supercomputer (DAS-2), a wide-area computer which consists of clusters of workstations distributed across the Netherlands.

The DAS-2 machine is used for research on parallel and distributed computing by five Dutch universities: University of Amsterdam, Vrije Universiteit Amsterdam, Delft University of Technology, Leiden University, and University of Utrecht. The cluster at the Vrije Universiteit contains 72 nodes while the other four clusters have 32 nodes. Each node contains two 1-GHz Pentium-IIIs, at least 1 GB RAM, a 20 GB local IDE disk (80 GB for Leiden and UvA), a Myrinet interface card, a Fast Ethernet interface. The nodes within a local cluster are connected by a Myrinet-2000 network, which is used as high-speed interconnect. In addition, Fast Ethernet is used as OS network. The five local clusters are connected by Surfnet, the Dutch university Internet backbone for wide-area communication.

The MPI implementation that is used in a Globus environment is MPICH-G2, which is a grid-enabled implementation of the MPI v1.1 standard. That is, using services from the Globus Toolkit, MPICH-G2 allows you to couple multiple machines, potentially of different architectures, to run MPI applications. MPICH-G2 automatically converts data in messages sent between machines of different architectures and supports multi-protocol communication by automatically selecting TCP for inter-machine messaging and vendor-supplied MPI for intra-machine messaging. The version available on DAS-2 is MPICH-GM, which uses Myricom's GM as its message passing layer on Myrinet. MPICH-GM is based on the MPICH package from Argonne/MSU. The current version is now able to use the fast local DAS-2 interconnect (Myrinet) on the local clusters; only communication between clusters goes over TCP/IP sockets.

3.2 The CrossGrid Testbed

For our more widely distributed set of experiments, we have used the CrossGrid pan-European distributed testbed. This infrastructure combines resources across 16 European sites (Fig. 1) into a large Grid Virtual Organization. The sites range from relatively small computing facilities in universities to large research computing centers, offering a heterogeneous set of resources to test the possibilities of a widely distributed experimental Grid framework. National research

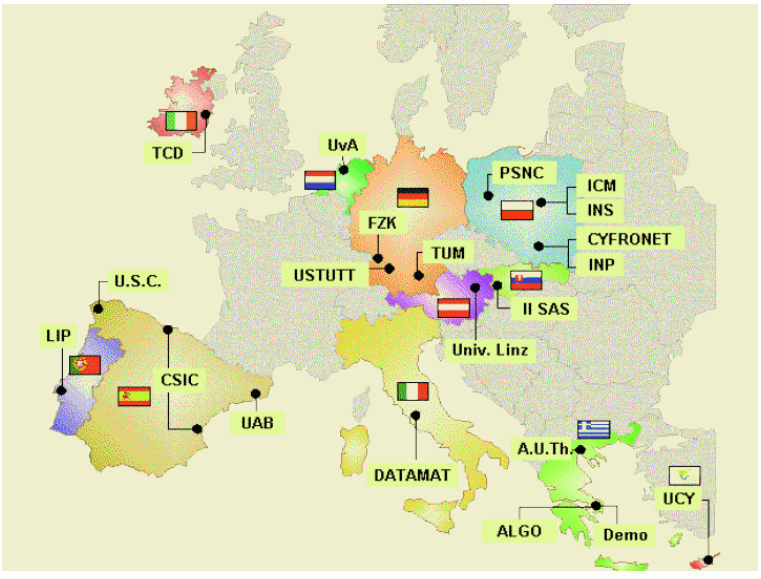


Fig. 1. Different locations in Europe participating in the CrossGrid network

networks and the high-performance European network, Geant, assure interconnectivity between all sites. The network includes a local step, typically inside a research center or university via Fast or Gigabit Ethernet, a jump via a national network provider at speeds that will range from 34 Mbits/s to 622 Mbits/s or even Gigabit, and a link to the Geant European network at 155 Mbits/s to 2.5 Gbits/s.

The CrossGrid team focuses on the development of Grid middleware components, tools and applications with a special focus on parallel and interactive computing, deployed across 11 countries. The added value of this project consists in the extension of the Grid to interactive applications. Interaction, in this context, refers to the presence of a human in a processing loop, and a requirement for near real-time response from the computer system. The CrossGrid testbed largely benefits from the EDG (Foster, Kesselman & Tuecke, [3]) experience on testbed setup and Globus (Karonis, Toonen & Foster, [6]) middleware distributions. The efforts to establish an integrated CrossGrid testbed started with the release of EDG 1.2.0; currently LCG2 is deployed in the testbed.

The CrossGrid testbed architecture and minimum hardware requirements are modeled after the LCG2 specification, with each site offering at least five system components:

- a gatekeeper that provides the gateway through which jobs are submitted to local farm nodes.
- a set of worker nodes or local farm computing nodes where jobs are actually executed; the combination of a gatekeeper with its worker nodes is usually called a computing element.

- a storage element or storage resource that includes a Grid interface ranging from large hierarchical storage management systems to disk pools.
- a user interface machine, used by end-users to submit jobs to the Grid computing elements.
- and a local configuration server, used to install, configure and maintain the above systems from a single management system.

Table 1. Sample Globus Resource Specification Language script for submitting MPI (MPICH-G2 device) jobs via Globus standard libraries, and performing file transfers and file access by Globus access to secondary storage. In the first line of the script we submit a job to a cluster in Spain requesting one processor. In the next 7 lines we set up the local environment and invoke the code called `nbodygrid` to run with N=65536 particles for one N-body time-unit. In the subsequent blocks of lines we simultaneously request one processor per cluster in Germany, Portugal and Cyprus, respectively

```
(&(resourceManagerContact="ce.grid.cesga.es:2119/jobmanager-pbs")
      (count=1)
      (label="subjob 0")(environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
      (POWER 1200)
      (DATADIR /home/cg013)
      (LD_LIBRARY_PATH /opt/globus/lib:/opt/edg/lib:/usr/local/lib)
      (GLOBUS_GRAM_JOB_CONTACT ce.grid.cesga.es:2119/jobmanager-pbs))
      (directory="/home/cg013/nbody_code/crossgrid/65536")
      (executable="/home/cg013/nbody_code/crossgrid/65536/nbodygrid")
      (arguments= "-n" "65536" "-t" "1")
)
(&(resourceManagerContact="ce010.fzk.de:2119/jobmanager-pbs")
      (count=1)
      .
      .
      .
      (arguments= "-n" "65536" "-t" "1")
      )
(&(resourceManagerContact="ce02.lip.pt:2119/jobmanager-pbs")
      (count=1)
      .
      .
      .
      (arguments= "-n" "65536" "-t" "1")
)
(&(resourceManagerContact="ce001.grid.ucy.ac.cy:2119/jobmanager-pbs")
      (count=1)
      .
      .
      .
      (arguments= "-n" "65536" "-t" "1")
)
```

The CrossGrid testbed includes a set of tools and services such as monitoring tools, development tools, a remote access server, portals and a prototype of the parallel resource broker.

Since the support of the CrossGrid resource broker for parallel applications using the MPICHG2 device was still being deployed at the time of our experiments, our job submissions were performed using the Globus job submission capabilities directly. As was the case for our experiments in the DAS2 network, the MPI package used for the tests was MPICH-G2, to allow for submission of a simulation job to a number of different sites, using different distributed processor topologies per run. Table 1 reports a sample job submission script for one run requiring 4 processors distributed among 4 different clusters.

4 Performance Results

The performance of a parallel code depends on the properties of the code itself, like the parallelization scheme and the intrinsic degree of parallelism, and on the properties of the parallel computer used for the computation. The main factors determining the general performance are the calculation speed of each node, the bandwidth of the inter-processor communication, and the network latency. In the case of a computational grid, the latency between different clusters may sensibly affect the execution times. To measure the effect of latency we performed test runs on the DAS-2 supercomputer and the CrossGrid testbed using the direct N-body code described in Sec. 2. We evolved the same initial configuration for one N-body time-unit[4] (Heggie & Mathieu, [5]), using 4 processors.

The total execution time is plotted in Fig. 2 as a function of the number of different locations hosting computing nodes. The low latency network on the DAS-2 generally results in good performance even if the nodes are allocated in different clusters. Only in the case of a very small number of particles, like for the $N = 4096$ run, the execution time increases steadily with the number of locations. This is due to an unfavorable computation to communication ratio for small N. The effects of inter-process communication are more evident for the CrossGrid runs, where the execution time generally increases with the number of locations. For large systems, however, the total time is dominated by the computation and the performance on the CrossGrid is comparable to that on DAS-2.

5 Summary and Conclusions

We have performed tests on two computational grids, the Dutch DAS-2 and the pan-European CrossGrid infrastructure. The application under consideration is a direct gravitational N-body code for simulating astrophysical stellar systems, like planetary systems, star clusters or dwarf galaxies. N-body methods are not

[4] A quantity proportional to the time needed for a typical star to cross the system.

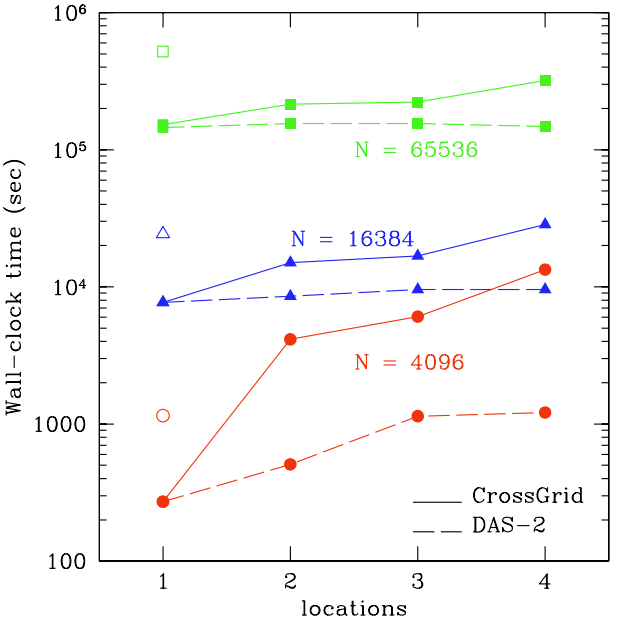


Fig. 2. Performance comparison of a direct gravitational N-body code on the DAS-2 wide-area supercomputer (dashed lines) and the CrossGrid distributed testbed (solid lines). The full dots refer to runs with N=4096, the full triangles to runs with N=16384 and the full squares to runs with N=65536. As a comparison, the empty symbols indicate the timing results in the case of a single processor on DAS-2. For all the runs we allocated 4 processors, distributed in 1 to 4 different locations. For example, in the case of one location all the 4 processors were selected in the same cluster, for two locations the processors were selected either two per location or three in one location and one in another, for four locations one processor per location was selected. The data related to one location were obtained on the DAS-2 only, which is effectively part of the CrossGrid

only applied to gravitational systems but can be used to efficiently solve a wide range of scientific problems ranging from the atomic to the cosmological scale. Applications include the study of equilibrium and non-equilibrium phenomena of microscopic and macroscopic molecular systems, equations of state and fluid dynamics. The algorithm is parallelized using a systolic scheme by means of the MPI library. For both grids we have allocated computing nodes in different locations, that is among different clusters participating in the grid network. The timing results indicate that the effects of latency are more prominent on the CrossGrid than on the DAS-2, as, for the latter, the clusters are interconnected with a faster and lower latency network. For both grids the communication effects on the performance decrease as the number of simulated particles increase. For large systems the total execution time is dominated by the computation rather than the communication and the load balance among the nodes is higher.

Acknowledgments

This work was supported by the Netherlands Organization for Scientific Research (NWO), the Royal Netherlands Academy of Arts and Sciences (KNAW) and the Netherlands Research School for Astronomy (NOVA). The authors acknowledge the DAS-2 and CrossGrid projects, for their help and support.

References

Dorband, E.N., Hemsendorf M., Merritt D.: Systolic and hyper-systolic algorithms for the gravitational N-body problem, with an application to Brownian motion. Journal of Computational Physics (2003), 185, 484-511

Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International J. Supercomputer Applications, (1997), 11(2):115-128

Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, (2001), 15(3)

Gualandris, A., Portegies Zwart, S., Tirado-Ramos, A.: Performance analysis of parallel N-body algorithms on highly distributed systems. Submitted to IEEE Transactions on Parallel and Distributed Systems

Heggie, D.C. and Mathieu, R.D.: Standardised Units and Time Scales The Use of Supercomputers in Stellar Dynamics, (1986), 267

Karonis, N., Toonen, B., Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Journal of Parallel and Distributed Computing (2003)

Makino, J., Hut, P.: Performance analysis of direct N-body calculations. ApJS (1988) 833–856

Makino, J.: A Modified Aarseth Code for GRAPE and Vector Processors. PASJ (1991) 859-876

Makino, J., Aarseth, S.J.: On a Hermite integrator with Ahmad-Cohen scheme for gravitational many-body problems. PASJ (1992), 44, 141-151

Introducing Grid Speedup Γ: A Scalability Metric for Parallel Applications on the Grid

Alfons G. Hoekstra and Peter M.A. Sloot

Section Computational Science, Laboratory for Computing,
Systems Architecture and Programming, Faculty of Science,
University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{alfons, sloot}@science.uva.nl
http://www.science.uva.nl/research/scs/

Abstract. We introduce the concept of Grid Speedup as a scalability metric for parallel applications on the grid and analyze it theoretically. It is shown that classes of applications exist for which good grid speedups can be expected. The class of stencil-based applications is taken as an example. The Grid computing community is challenged to demonstrate large grid speedups on real Grid computing environments.

1 Introduction

We consider 'traditional' tightly coupled parallel applications in Grid Computing environments. This is the *Computation-Centric* class of Grid application [1] which, according to Allen et al. "turned to parallel computing to overcome the limitations of a single processor, and many of them will turn to Grid computing to overcome the limitations of a parallel computer." However, common wisdom is that running a tightly coupled parallel application in a computational grid is of no general use because of the large overheads that will be induced by communications between computing elements (see e.g. [2]) and the inherent unreliable nature of a computational Grid.

We introduce the concept of Grid Speedup and analyze it theoretically on a Homogeneous Computational Grid. To do so we assume a two-level hierarchical decomposition. We will show that in terms of Grid Speedup good performance of Computation Centric Grid applications is possible, provided that workloads per Computing Element remain at a sufficiently high level. This large grain size demand is off course not very surprising and well known from parallel computing. Moreover, we introduce grid fractional overhead functions that are sources of grid efficiency reductions. Finally, we will consider one specific example and analyze for which problem sizes grid computing is beneficial.

This new scalability metric will be useful to predict performance of applications on a computational grid, and may be used in e.g. intelligent grid resource managers to facilitate more advanced selection of resources for applications requesting computational power from a Grid.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 245 – 254, 2005.

2 Notation

We use the notation of parallel work W_i as in [3-5]. So, W_i represents the amount of work (expressed e.g. in Mflop) with a degree of parallelism (DOP) i. Define Δ as the computing capacity of a processor (expressed in e.g. Mflop/s). The amount of work executed while running a part of the program with DOP = i is

$$W_i = \Delta i t_i , \tag{1}$$

where t_i is the total amount of time during which DOP = i. The total amount of work is

$$W = \sum_{i=1}^{m} W_i , \tag{2}$$

with m the maximum DOP in the application. Now assume that the workload W is executed on p processors. The execution time for the portion of work with DOP = i is

$$t_i(p) = \frac{W_i}{i\Delta}\left\lceil \frac{i}{p} \right\rceil . \tag{3}$$

Notice that in the formulation of Eq. [3], possible load imbalance is implicitly taken into account. The total execution time of workload W on p processors, $T_p(W)$, equals

$$T_p(W) = \sum_{i=1}^{m} t_i(p) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}\left\lceil \frac{i}{p} \right\rceil + Q(W, p) \tag{4}$$

with the (communication) overhead for a p processor system for completion of the workload W defined as $Q(W,p)$, and understanding that $Q(W,1) = 0$.

In this paper we assume the simplest possible workload: $W_i = 0$ if $i \neq p$, i.e. a fully parallel workload. In this case we find

$$T_1(W) = T_1(W_p) = \frac{W_p}{\Delta} , \; T_p(W) = \frac{W_p}{p\Delta} + Q(W_p, p) , \tag{5}$$

and from this we derive expressions for relative speedup S_p and efficiency ε_p

$$S_p = \frac{T_1(W)}{T_p(W)} = \frac{p}{1 + f(W_p, p)} \; ; \; \varepsilon_p = \frac{T_1(W)}{pT_p(W)} = \frac{1}{1 + f(W_p, p)} \tag{6}$$

with $f(W,p)$ the fractional overhead function

$$f(W, p) = p\Delta \frac{Q(W, p)}{W} . \tag{7}$$

3 Hierarchical Decomposition

We will now consider the case of a parallel application with a workload W running in a grid-computing environment. We assume that within the Virtual Organization (VO)

in which the application is running, the workload is decomposed among C Computing Elements (CE's), and that each CE in itself is some HPC system (typically a parallel computer with p nodes). Such hierarchical resource models were already proposed in [6,7] So, we see a two-level hierarchical decomposition appearing. First, the workload is decomposed between C CE's, and next within each CE the portion of workload is again decomposed between the processors within a specific CE (see Fig. 1). Examples of Grid Enabled parallel applications applying such hierarchical decompositions can be found in the recent literature, e.g. [8].

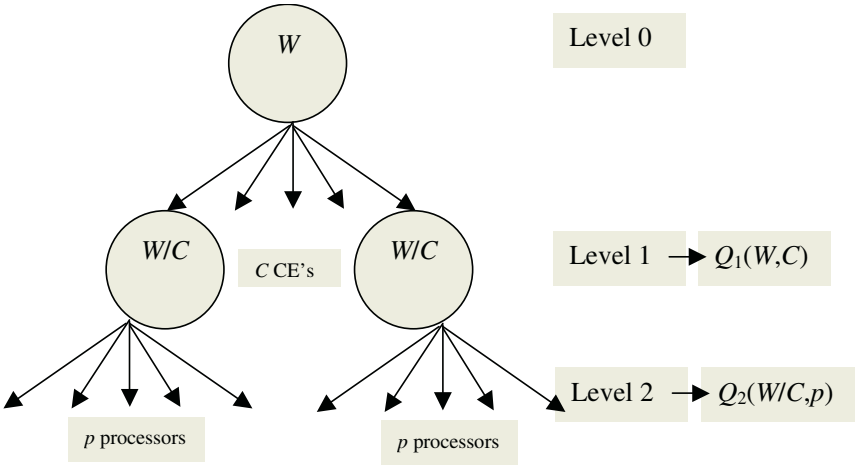


Fig. 1. The hierarchical decomposition

Referring to Fig. 1, we identify three levels. This first is level 0, which is the full non-decomposed workload W. This level would be the sequential computing level. The second level is level-1 decomposition, i.e. the division of the workload between the CE's. This decomposition induces a level-1 overhead $Q_1(W,C)$, e.g. communication between CE's. Next, within each CE the workload W/C is again decomposed between the p processors of the CE. On this level we again encounter an overhead $Q_2(W/C,p)$. If we have one CE (i.e. $C = 1$ and understanding that $Q_1(W,1) = 0$) we return to the standard parallel case where $Q_2(W,p)$ now plays the role of $Q(W,p)$ of the previous section.[1]

Please note that by now we have made the implicit assumption that each CE receives an equal share of the total workload, and that each CE has the same amount of processors. We will work under this assumption. Moreover, we will assume that each processor in each CE has the same computing capacity Δ, and that within each CE the same overhead function $Q_2(W,p)$ applies. We refer to this situation as a *Homogeneous Computational Grid* (HCG). The formalism can be extended to Heterogeneous systems in a straightforward way, using e.g. concepts from [9].

[1] Clearly we can write $Q_2(W,p) = Q(W,p)$.

4 Grid Speedup

We denote the execution time on a HCG with C CE's and p processors per CE as $T_{C,p}(W)$, and with the definitions as introduced earlier we find

$$T_{C,p}(W) = \frac{W_p}{pC\Delta} + Q_2(W_p/C, p) + Q_1(W_p, C) \tag{8}$$

The question now is how to interpret Eq. 8. As already noted in Section 1, common wisdom is that running a application decomposed over several CE's is not very usefull because of the large overheads that will be induced by the communication between the CE's. A first look at Eq. 8 confirms this, the extra overhead expressed by $Q_1(W_p,C)$ contains it all. However, by running the application on more than one CE, we also have more processors to do all the work (a factor C more, reducing the time for pure computation by a factor C). Moreover, the overhead per CE is changed, from $Q_2(W_p,p)$ in the case of running on one CE to $Q_2(W_p/C,p)$ in the case of running on C CE's. Although we do not know this function without turning to specific applications, at this point it is clear that a more detailed analysis is needed.

One can now easily compute the parallel speedup of the application on the HCG. One can however question to what extend this speedup is a good metric to assess the added value of decomposing an application over CE's. Let us recall the reason why parallelism was introduced. Researchers wanted to use parallel computers for two reasons.

1. They were compute bound, meaning that on one processor the computing time was unacceptably high, and more computing power was needed.
2. They were memory bound, meaning that the memory consumption of the application was so large that it would not fit in memory of a single processor. Using more processors (assuming distributed memory computers here) would not only increase the computational power, but also the amount of available memory.

In many cases both reasons applied. To assess the quality of the parallel application, the speedup/efficiency metric was applied. Also, scaled speedup models were introduced, to cope with the fact that researchers will immediately increase the amount of computational work once they get more computing power and available memory.

The bottom line of the previous discussion is that in order to analyze the added value of parallelism, one compared the execution time of the parallel application with a reference value: the execution time on a sequential computer. Let us now analyze performance in a grid-computing environment. We can argue that our reference value in this case should not be the single processor, but the execution time on one single CE. The reason that we decide to decompose our application over more that one CE are exactly the same as our original reasons to parallelize our application, we are compute bound or memory bound in one CE, or a combination of both. So, the question that we must ask ourselves is: "does decomposing over C CE's give us any added value *as compared to* running on one CE?". This leads us to the concept of *Grid Speedup*, defined as

$$\Gamma_p^C = \frac{T_{1,p}(W)}{T_{C,p}(W)}. \tag{9}$$

So, in Grid Speedup we take the quotient of the execution time of our application on 1 CE and the execution time on C CE's. If $p = 1$ we are back in the normal situation of a parallel computation, with C now playing the role of the number of processor. Also note that Grid speedup depends on two parameters, the number of CE's and the number of processors per CE. Grid efficiency is now defined as

$$\gamma_p^C = \frac{T_{1,p}(W)}{CT_{C,p}(W)}. \tag{10}$$

Let us compute the grid speedup for the example of a single parallel workload W_p. Substitute Eq. 8 into Eq. 10, which after some algebra results in

$$\Gamma_p^C = \frac{C}{1 + g_2(W_p, p, C) + g_1(W_p, p, C)}. \tag{11}$$

We defined two fractional grid overhead functions:

$$g_1(W_p, p, C) = C\frac{Q_1(W_p, C)}{T_{1,p}(W_p)}; \tag{12}$$

$$g_2(W_p, p, C) = \frac{CQ_2(W_p / C, p) - Q_2(W_p, p)}{T_{1,p}(W_p)}. \tag{13}$$

The first fractional grid overhead function g_1 plays exactly the same role as the fractional overhead f (Eq. 7) in the simple parallel case. This analogy becomes clear by realizing that f can be rewritten as $f = pQ(W_p,p)/T_1(W_p)$. So, we can obtain good grid speedups if the grid fractional overhead g_1 is small, that is, if we let the grain size, defined as the portion of work per CE, be large enough such that the amount of work per CE is much larger than the amount of overhead induced by the level-1 decomposition Q_1.

The hierarchical decomposition introduces another fractional grid overhead g_2, which expresses the relative difference in overhead inside a CE between the level-1 decomposed workload W_p/p and the original workload W_p. For the special case that

$$CQ_2(W/C, p) = Q_2(W, p) \tag{14}$$

we immediately find that $g_2 = 0$ and

$$\Gamma_p^C = \frac{C}{1 + g_1(W_p, p, C)}. \tag{15}$$

5 An Example: Stencil-Based Operations

Let us consider a prototypical scientific computation: that of a stencil based operation. So, we assume some computational mesh, and we assume an iterative procedure on

this mesh where during each iteration each mesh point is updated, using information from a small local neighborhood of that point. This could e.g. be a time dependent explicit finite difference, finite volume or finite element computation, a Cellular Automaton, or a pixel based image analysis algorithms. In the sequel we assume a two-dimensional square Cartesian mesh with $n \times n$ points.

First consider the case where we apply a 1 dimensional 'strip wise' decomposition of the computational domain. This is schematically drawn in Fig. 2. The left panel shows the decomposition in the normal parallel case (i.e. $C = 1$), and the right panel shows the hierarchical decomposition. Note that in the hierarchical decomposition we choose to decompose the computational domain in each CE along the dimension with the shortest size.

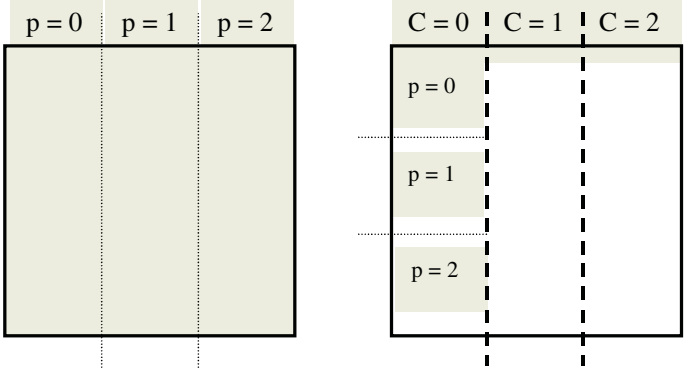


Fig. 2. Decomposition in case 1, strip wise. The left panel shows the decomposition in the normal parallel case, the right panel shows the hierarchical decomposition. In this figure we assume $C = p = 3$

The execution time on one CE in this case becomes

$$T_{1,p} = \frac{n^2}{p\Delta} + 2n\tau_{com}, \tag{16}$$

where τ_{com} is the communication time needed to send the stencil information of a point on the boundary of the processor domain to a neighboring processor. The factor 2 emerges because each processor should communicate with its left and right neighbor.[2] On more than one CE the hierarchical decomposition results in the following execution time:

$$T_{C,p} = \frac{n^2}{pC\Delta} + 2n\tau_{grid} + 2\frac{n}{C}\tau_{com}. \tag{17}$$

Here, τ_{grid} is the communication time needed to send the stencil information of a point on the boundary of a CE to a neighboring CE. Note that we assume here a

[2] For 2 processors this could change from a factor 2 to a factor 1 if the algorithm does not assume period boundary conditions. We neglect this here.

communication pattern where we first communicate between CE's (giving rise to the term $2n\tau_{grid}$) followed by a communication step inside each CE (the $2n/C\ \tau_{com}$ term).

Inspection of Eq. 16 and 17 shows that we can write for the overhead functions

$$Q_1(W, C) = 2n\tau_{grid} \tag{18}$$

$$Q_2(W/C, p) = 2\frac{n}{C}\tau_{comm} \tag{19}$$

In this example $CQ_2(W/C, p) = Q_2(W, p)$ applies, so we find immediately for the Grid speedup, using Eq. 15,

$$\Gamma_p^C = \frac{C}{1 + \dfrac{2Cn\tau_{grid}}{\dfrac{n^2}{p\Delta} + 2n\tau_{com}}}, \tag{20}$$

which we rewrite to

$$\Gamma_p^C = \frac{C}{1 + \dfrac{2C\alpha}{\beta + 2}}, \tag{21}$$

with

$$\alpha = \frac{\tau_{grid}}{\tau_{com}}, \tag{22}$$

$$\beta = \frac{n}{p\Delta\tau_{com}}. \tag{23}$$

Note that dimensionless parameter α expresses the imbalance between inter- and intra CE communication. The dimensionless number β contains the grain size of the application running on 1 CE, and the balance between computational speed and communication within one CE. Note that if one would compute the fractional communication overhead in this example (Eq. 7) one would find that $f = 2/\beta$. Large grid speedups are obtained if α is very small and/or β is very large.

We should explicitly analyze the case of $C = 2$ and assuming no periodic boundary conditions in our application. In that case we have

$$Q_1(W, C = 2) = n\tau_{grid} \tag{24}$$

and Eq. 21 changes to

$$\Gamma_p^{C=2} = \frac{2}{1 + \dfrac{2\alpha}{\beta + 2}}. \tag{25}$$

The first question one could ask is: "when does it pay off to execute my application on more than one CE?". Translated in our formalism, when will $\Gamma_p^C \geq 1$. Clearly, a

speedup marginally larger then 1 is probably not worth the effort of porting an application to the grid. A better analysis would be to demand that the Grid efficiency should be larger than a certain value, say larger then γ_0. So, we demand that $\Gamma_p^C / C \geq \gamma_0$. This results in

$$\beta \geq \begin{cases} \dfrac{2\gamma_0\alpha}{1-\gamma_0} - 2 \;\text{ if } C = 2 \textit{ and} \text{ no periodic boundaries apply} \\ \dfrac{2\gamma_0\alpha C}{1-\gamma_0} - 2 \;\text{ otherwise} \end{cases} \tag{26}$$

Note that if we put $\gamma_0 = 1/C$ we return to demanding that the grid speedup should be larger than 1.

Let us now try to find some representative numbers for the parameters, and compute grain sizes for which we expect to see a beneficial effect of using a Grid for tightly coupled applications.

As a representative application that falls in the category of the stencil based operations as analyzed in this section we choose the Lattice Boltzmann Method [10]. As was recently reported by Wellein et al. during ICMMES 2004 in Braunschweig [11] a Lattice Boltzmann simulation attains computational speeds in the order of 1 Gflop/s on modern processors such as a 1.3 GHz Itanium2 or a 2.4 GHz Opteron. Updating one lattice point requires in the order of 200 floating point operations, so we find $\Delta = 1\times10^9 / 2\times10^2 = 5\times10^6$ sites/s. Now let us assume that $\alpha = 10$, i.e. inter CE communication is 10 times faster then intra CE communication (see e.g. discussion in [12]). Finally, assume that processors in a CE can communicate at a speed of 10 Mbyte/s and that in a 2D Lattice Boltzmann simulation 4 floating point numbers must be sent in both directions. Under these assumption we find $\tau_{\text{comm}} = 4\times8 / 10^7 \sim 3\times10^{-6}$.

Let us first consider the case of two CE's. In this case the result is that (see Eq. 26, using $\alpha = 10$)

$$\beta \geq \frac{20\gamma_0}{1-\gamma_0} - 2\,. \tag{27}$$

The breakeven point is for $\gamma_0 = 1/C = 0.5$, so $\beta \geq 18$. Using Eq. 23 we find

$$\frac{n}{p} \geq 18\Delta\tau_{comm}\,. \tag{28}$$

With the numbers as defined above Eq. 28 results in $n/p \geq 270$. So, only if $n \geq 270\times p$ the Grid Speedup will be larger than 1 and therefore the execution time of running the simulation on 2 CE's will be smaller than running on 1 CE. Now suppose we want to have a grid efficiency of at least 0.8 (i.e. a grid speedup of 1.6 on 2 CE's). In that case we find $n/p \geq 1170$. We clearly see that in order to get real benefits of running this stencil-based operation on a Computational grid we need very large domains. Assuming e.g $p = 10$ (which would be a relatively small parallel computer) boils down to demanding that in order to pay off, $n \geq 2700$, and in order to get real

benefit $n \geq 11700$. The positive conclusion is that albeit large, these are not unrealistically large grain sizes and that such applications can truly achieve grid speedup on 2 CE's. Clearly for $C > 2$ larger dimensions are needed, and our analysis gives a quick estimate of what to expect.

6 Discussion and Conclusions

We have introduced a new metric, Grid Speedup Γ, that allows analyzing the performance of tightly coupled parallel applications on a Homogeneous Computational Grid. Using the concept of a two level hierarchical decomposition of the workload, a general formalism was introduced that allows computing Grid Speedup in terms of two fractional overhead functions. Using this formalism we analyzed in detail a prototypical application based on a two-dimensional stencil operation. In turns out that two dimensionless numbers now determine Grid Speedup. The number α expresses the ratio of inter- and intra CE communication and the other, β, is inversely proportional to the well-known fractional communication overhead of the parallel application. Estimating the parameters appearing in the model leads to the conclusion that for a small number of CE's good Grid Speedups is attainable, as long as the work per CE is large enough.

More extended analysis then the one presented here clearly is possible. For instance, in our example a more efficient decomposition, in which the intra- and inter CE decomposition run in the same direction, can be used, leading to the count intuitive conclusion that for such decomposition the Grid Speedup can be larger due to a latency hiding effect. A more realistic case in which CE's have different number of processors, leading to the concept of a fractional C, could be discussed. Or, more realistic workloads, including a sequential workload W_1 could be considered. We will analyze all this in more detail in a future paper.

We must now test these theoretical ideas and try to measure Grid Speedups on real Computational Grids. It will be quite challenging to try to apply our theoretical ideas to real Grids, that will behave must more dynamic and erratic as our ideal Homogeneous Computational Grid. Fortunately test systems, such as the Dutch ASCI – DAS2 computer [13], that come close to the idea of a Homogenous Computational Grid are available as a first test site, before embarking on scalability measurement on real Grids, such as the European CrossGrid test bed [14].

References

1. Allen, G., Goodale, T., Russell, M., Seidel, E., Shalf, J.: Classifying and Enabling Grid Applications. In Berman, F., Fox, G.C., Hey, A.J.G. (Eds.); Grid Computing, Making the Global Infrastructure a Reality. Wiley, chapter 23 (2003)
2. Lee, C., Talia, D.: Grid Programming Models: Current Tools, Issues and Directions. In Berman, F., Fox, G.C., Hey, A.J.G. (Eds.): Grid Computing, Making the Global Infrastructure a Reality. Wiley, chapter 21, specifically section 21.2.3 (2003)
3. Hwang, K.: Advanced Computer Architecture, Parallelism, Scalability, Programmability. McGraw-Hill, New York, chapter 3 (1993)

4. Kumar, V., Gupta, A.: Analyzing Scalability of Parallel Algoritms and Architectures. J. Parallel Distrib. Computing 22 (1994) 379-391
5. Sun, X.H., Ni, L.M.: Scalable Problems and Memory-Bounded Speedup. J. Par. Distr. Comp. 19 (1993) 27-37
6. Halderen, A.W. van, Overeinder, B.J. Sloot, P.M.A., Dantzig, R. van, Epema, D.H.J, Livny, M.: Hierarchical Resource Management in the Polder Metacomputing Initiative. Parallel Computing 24 (1998) 1807-1825
7. Iskra, K.A., Belleman, R.G., Albada, G.D. van, Santoso, J., Sloot, P.M.A., Bal, H.E., Spoelder, H.J.W., Bubak, M.: The Polder Computing Environment, a system for interactive distributed simulation. Concurrency and Computation: Practice and Experience 14 (2002) 1313-1335
8. Keller, R., Gabriel, E., Krammer, B., Müller, M., Resch, M.M: Towards Efficient Execution of MPI Applications on the Grid: Porting and Optimization Issues. J. Grid Comp. 1 (2003) 133-149
9. Bosque, J.L., Pastor, L: Theoretical analysis of scalability on heterogeneous clusters. In proceedings of the 4th IEEE/ACM International Conference on Cluster Computing and Grids, Chicago (published on CD), April 2004.
10. Succi, S: The Lattce Boltzmann Equation for fluid dynamics and beyond. Oxford Science Publications (2001)
11. Wellein, G. et all.: Optimization Approaches and Peformance Characteristics of Lattice Boltzmann Kernels. Presented during International Conference for Mesoscopic Methods in Engineering and Science, Braunschweig, July 2004. To be published in Computers and Fluids.
12. Berman, F., Fox, G., Hey, T.: The Grid: past present future. in Berman, F., Fox, G.C., Hey, A.J.G. (Eds.): Grid Computing, Making the Global Infrastructure a Reality. Wiley, chapter 1 (2003)
13. Bal, H.E. et all.: The Distributed ASCI supercomputer project: Operating Systems Review 34 (2000) 76-96
14. Gomes, J. et all.: First prototype of the CrossGrid Testbed. in Rivera, F.F., Bubak, M., Tato, A.G., Doallo, R. (Eds.): Grid Computing, Lecture Notes in Computer Science, Vol. 2970, . Springer-Verlag, Berlin Heidelberg New York (2004) 67-77

A Dynamic Key Infrastructure for GRID

H.W. Lim and M.J.B. Robshaw

Information Security Group,
Royal Holloway, University of London,
Egham, Surrey TW20 0EX, UK
{h.lim, m.robshaw}@rhul.ac.uk

Abstract. This paper introduces the concept of a dynamic key infrastructure for GRID. It utilises the properties of Identity-based Cryptography (IBC) to simplify key management techniques used in current Public Key Infrastructure (PKI) settings for GRID. This approach can offer greater simplicity, flexibility, and enhanced computation trade-offs.

1 Introduction

GRID computing is seen as the next generation computing technology, offering virtually "unlimited" resource sharing for computationally intensive, and advanced science and engineering problems. It also provides a better opportunity for improving life through "unlimited" access to all kinds of knowledge pools and utilising many applications and different devices to support daily activities. After close to a decade of research and development, it is believed that the GRID vision is feasible based on a combination of technology trends and research advances [5].

Currently, Public Key Infrastructure (PKI) is the most widely used security infrastructure for GRID implementations [11]. It is an interesting exercise to consider whether PKI-based GRID security solutions are entirely matched to such a dynamic, adaptable, and scalable environment. In an implementation based on Globus Toolkit's Grid Security Infrastructure (GSI) [6, 8, 14], two type of certificates are used: public key certificate and proxy certificate. The former is required for entity authentication and the latter is used for single sign-on and rights delegation [12, 13]. One important motivation for the proxy certificate is to limit the exposure of long-term credentials (which normally need to be updated yearly) by using proxy credentials with much shorter lifetime (typically on the order of hours or days) in a job submission. The heavy use of proxy credentials involving creation of proxy certificates, computation of short-lived public/private key pairs, and verification of the proxy certificates, is not trivial.

In this paper, we propose a dynamic key infrastructure for GRID (DKIG), a lightweight and scalable security infrastructure through the use of properties from Identity-based Cryptography (IBC) [2, 10]. We introduce the concept of a *master-public-key mould* which can be used by any user to compute a per-session public key on-the-fly. The corresponding session private key can be extracted

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 255–264, 2005.

easily by the key mould owner with his master secret key. A master-public-key mould is a permanent credential of a user requiring no update unless the associated master secret key is compromised. By replacing certificates in the current PKI setting with these master-public-key moulds (which need to be initially authenticated by a trusted authority), we achieve similar security goals as in the current PKI setting in a more elegant manner. We also present IBC-based authenticated key establishment and rights delegation protocols that use ephemeral public/private key pairs, which can be bound to a specific job submission. This approach, which is difficult to emulate with current PKI techniques, seems to offer the potential for greater flexibility with less computation cost.

The remainder of this paper is organised as follows. Section 2 gives some background information on PKI-based security infrastructure, and an IBC-based security solution. In Section 3 we propose our DKIG model with considerations of its fundamental framework and security mechanisms. Then Section 4 presents some insights on how the Transport Layer Security protocol might be used in the new framework. In Section 5, we will discuss the computation trade-offs of delegation protocols in both GSI and DKIG settings. Section 6 concludes the paper.

2 Background

We briefly survey Globus Toolkit's Grid Security Infrastructure (GSI), the most prevailing security infrastructure thus far implemented in Grid projects [11], and a proposed alternative approach using Identity-based Cryptography [7].

2.1 PKI and Globus Toolkit's GSI

PKI (in this paper we assume X.509-based PKI) plays a vital role in facilitating key distribution service to its users. The practicality of a public key cryptosystem relies on the assurance of the authenticity of the public keys of the users. All public keys generated by users or the Certificate Authority (CA) are random arbitrary strings. These strings have no relation with the users' Distinguished Name (DN) and X.509 public key certificates are required to bind these unknown strings to some easily recognised user identities.

One early development of PKI-based security infrastructure for Grid was Globus Toolkit's GSI. In GSI, an X.509 public key certificate is issued and signed by a Grid CA. The X.509 certificate is used to support authentication in a protocol like Transport Layer Security (TLS), which establishes secure communication channels between entities. Depending on the Grid CA's policy, a user's long-term public/private key pair and its associated certificate are usually updated and renewed yearly.

The proxy certificate, a variant of the X.509 certificate, was introduced to enable single sign-on and rights delegation. It is a short-lived certificate and typically carries restricted credentials of the user and expires upon completion of a job. Before a user submits a job request, he must create a proxy certificate which includes generating a new public/private key pair and signing the proxy

certificate with his long-term private key. This newly created proxy certificate can then be used for repeated authentication with other GRID entities without accessing the user's long-term private key. For rights delegation, e.g. if A wants to delegate her privileges to a target service provider X, three steps are required: (i) X creates a new public/private key pair and send a signed request (with the new private key) to A; (ii) A verifies the new public key, creates a new proxy certificate and signs it with her current proxy credential (short-lived private key); and (iii) A forwards the new proxy certificate to X and thus delegates her rights to X [13]. Nevertheless, the creation of proxy certificates can be computationally expensive. In both single sign-on and delegation services, new key pairs generation are required. Furthermore, additional signature generation and verification operations are needed in the delegation process.

2.2 Alternative Security Infrastructure

While there are many alternative approaches in designing security infrastructures for GRID, we are particularly interested in investigating the potential of IBC in GRID environments.

In 1984, Shamir [10] introduced the concept of IBC in which the public key can be generated from a publicly identifiable information such as a person's name, e-mail address, or IP address. The corresponding private key is generated and maintained by a Private Key Generator (PKG) which holds a master key. However, it was not until 2001 that new wave of developments took place. Then Boneh and Franklin [2] successfully proposed a fully functional and practical Identity-based Encryption (IBE) scheme making use of *bilinear maps* (or pairings) between groups.

To simplify, the following four algorithms underpin the Boneh and Franklin IBE scheme.

- Setup: Given a security parameter, the PKG generates params (system or public parameters) which will be made publicly known, and a master key which will be kept secret.
- Extract: This algorithm is run by the PKG to extract a private key from a given public key. It takes params, master key, and an arbitrary identifier ID (public key string) as input, and returns a private key.
- Encrypt: This algorithm uses params and ID to encrypt a message and generate a ciphertext.
- Decrypt: By using params and a private key as the input, the algorithm returns a plaintext from a ciphertext.

Suppose that Alice wants to send a message M secretly to Bob using an identity-based cryptosystem. She does not need to verify the authenticity of Bob's public key by retrieving Bob's public key certificate (which happens in a conventional PKI-based system). Instead Alice simply encrypts the message with Bob's ID, e.g.'`bob@xyz.com`'. Clearly, Alice needs to know the params of Bob's PKG. If Bob does not already possess the corresponding private key d, he

has to obtain it from his PKG. If the PKG is satisfied that Bob is a legitimate receiver of the ciphertext C, the PKG takes its params, master key, and Bob's ID to generate d, which will then be used by Bob to decrypt C.

Lim and Robshaw [7] illustrated how IBC could be applied in a GRID environment. In their proposal, each *virtual organisation* (VO) has its own PKG and all the users of that VO share the same params (which has been certified by a Grid CA). When Alice wants to send a job request securely to Resource X, she can in principle encrypt the message with an identifier $\mathsf{ID} = X \parallel timestamp$. In addition, one can add more granularity to impose restrictions on the receiving party by concatenating the policy into the public key string. This offers greater flexibility to key generation by the sender.

However, there are two potential downsides to an IBC-based GRID infrastructure, namely, the need for a user to maintain an independent secure channel with the PKG for the retrieval of the private key; and the fact that the PKG knows all the private keys of its users which makes it vulnerable to single point of attack.

3 Dynamic Key Infrastructure for GRID

Here we describe a new dynamic key infrastructure for GRID (DKIG). This term is intended to capture the notion that the public keys used in the infrastructure change dynamically on a per-job-request basis. The DKIG model will be discussed at a relatively abstract level.

3.1 Motivation

We propose a different approach of simplifying key management issues mentioned in the previous section. In the conventional IBC-based setting, all the users within a domain (or VO) share the same params generated by the PKG. However, params can be turned into a master-public-key mould if each entity generates and publishes their own unique params. Our core idea features in a different guise in [2], where Boneh and Franklin proposed a means of delegating decryption keys by having a users play the role of the PKG by publishing his params to other users. In DKIG we extend this idea to a GRID environment whereby each user acts as his own PKG. Interestingly, this technique also seems to solve the key escrow and private key distribution problems in an IBC-based system. In addition, when compared to the current use of PKI in GSI, a user's params (master-public-key mould) can provide the functionalities of both the certificate and the proxy certificate. In other words, the user does no longer need to create new proxy certificates for job submissions and rights delegations.

3.2 Defining DKIG

We propose an environment whereby a user manipulates his params as he would a certificate. When a user A (Alice) wants to send a message securely to a resource X, she can derive a short-lived public key on-the-fly using X's ID and

params without having it certified as in [12, 13]. Obviously, the params must be verified and signed by a Grid CA *a priori* just like any other long-term public key certificate.

To avoid confusion between 'identity' and 'identifier'(ID), we use entity name EN to represent the identity of a VO member and refer to ID as the arbitrary identifying public key string used in the IBE scheme [2] (where ID contains EN). In our setting, the public key of X can be computed as $K_X = H(\mathsf{ID}_X)$ where $\mathsf{ID}_X = EN_X \parallel \langle optional \rangle$. Note that in general, H is a one-way hash function and $\langle optional \rangle$ denotes additional fields that can be included in ID such as IP address, random number, timestamp, and so on. However, the system must be aware of which specific additional fields might be used so that the receiver will be able to decrypt the message with the associated master key. For signing and verification algorithms, we refer to Cha and Cheon's Identity-based Signature (IBS) scheme [3] with similar conversion as in our encryption and decryption settings whereby each user has his own params.

The fundamental DKIG framework is developed by considering the need of a secure environment for job submissions from the users to remote resources. We introduce a trusted third party within a VO called an Administration Server (S) which acts like an authentication server, as well as an authorization server. This server keeps track of all the legitimate users and has access to a policy database that stores up-to-date access rights of each entity in the VO.

3.3 Security Mechanisms

In this section we illustrate some security goals required for GRID environments. At the initial setup phase, we assume that each new member will be given a copy of the public key certificate of the Grid CA and also the certified params of S (by the Grid CA). Also, these new members must generate their own params and get them validated by the Grid CA.

Authentication and Authorization. We present a protocol for authentication and authorization of job submissions by A to X via S, as shown in Protocol 1. After A has identified X as her target resource, she needs to send a job request J_{Req} to S to gain authorization for accessing X. J_{Req} contains ENs of the requestor and the target resource, their respective Administration Server, job description $Desc$, a random number r, and timestamp t, e.g. J_{Req} in Message (1) is $\{EN_A, EN_X, EN_S, Desc_A, r_A, t_A\}$. The random number r_A is used to detect replay attack and to ensure uniqueness of each ephemeral public key created by the user which can be tied down to a specific job submission. It might be replaced by a counter which may offer certain trade-offs.

Protocol 1 Authenticated key establishment for job request

(1) $A \rightarrow S: \; E_{K_S^*}[J_{Req}],\, r_A$
(2) $S \rightarrow A: \; E_{K_A^*}[Sig_{K_S^*}[J_{Req}],\, r_A]$
(3) $A \rightarrow X: E_{K_X^*}[J_{Req},\, Sig_{K_S^*}[J_{Req}],\, K_{AX}],\, r_A$
(4) $X \rightarrow A: \{r_A\}_{K_{AX}}$

where

$E_{K_A}[m]$ = identity-based encryption of data m using a public key K_A of entity A
$Sig_{K_A}[m]$ = identity-based signature with appendix of data m using a private key K_A of entity A
$\{m\}_{K_{AX}}$ = symmetric encryption of data m using a symmetric session key K_{AX} shared between entities A and X
J_{Req} = job request information
r = random number
$*$ = short-lived

The short-lived public key that A creates is based on the identifier $\mathsf{ID}_S^* = EN_S \,\|\, r_A$. Because of this, r_A is also attached in Message (1) in cleartext so that S knows the ID used for extracting the corresponding decrypting key. When S receives the message, it would decrypt it and match EN_A with the policy database. If the status is that A is an authorized user of X and the timestamp stated in the request is acceptable, then S would digitally sign J_{Req} with a private key that it extracts based on ID_S^*. If for some reason A is barred from accessing X, then it would reject A's request. In principle $Sig_{K_S^*}[J_{Req}]$ serves as an *access token* (access approval) of A. To reply to A as in Message (2), S encrypts the message with $\mathsf{ID}_A^* = EN_A \,\|\, r_A$. The next thing that A does is to forward a message that contains all the required information as stated in Message (3) to X. Upon receiving the message, X decrypts and verifies S's signature on the access token. X also compares the contents of the token with J_{Req} forwarded by A. In addition and more importantly, X validates A's access rights with its local policy set by a local administrator (note that local policy shall override VO policy enforced by S if there is a conflict). If all checks succeed, X replies to A with the session key to acknowledge mutual authentication and its readiness to establish a secure channel with A. As in a common key establishment protocol, the shared symmetric session key is used to protect data confidentiality and integrity.

Rights Delegation and Single Sign-on. Rights delegation is a core service provided by the Globus Toolkit's GSI through the proxy certificate [9, 13]. In DKIG, we design a simpler approach as shown in Protocol 2 to deliver similar services without the creation of "proxy params" (which would be analogous to the need for a proxy certificate in the PKI setting).

Protocol 2 Rights delegation for job process

Delegation from A to X

(1) $A \rightarrow X : Sig_{K_A^*}[EN_A, EN_X, r_A],\ r_A$

Delegation from A to X to Y

(2) $X \rightarrow Y : Sig_{K_X^*}[EN_Y,\ Sig_{K_A^*}[EN_A, EN_X, r_A]],\ r_A$

If A wishes to delegate her rights to X, in principle, she could sign a *delegation token* with her short-lived private key (where $\mathsf{ID}_A^* = EN_A \| r_A$) and forwards it to X. Message (1) shows that the delegation token is a string containing X's identity and the random number tied to a specific job request. A malicious user will not be able to intercept and exploit A's access rights unless he can forge A's signature. Thereafter, X can access other resources on A's behalf by presenting the delegation token. Any party can use params_A and ID_A^* (which are publicly available) to verify the delegation token. For instance, if X uses the token to access Z, Z can verify A's signature on the token and check if the token contains X's identity before allowing a data connection between X and Z. This approach cannot be achieved through current PKI settings because one needs to create a new "random looking" public key and then bind it with a proxy certificate. Should X need to further delegate A's access rights to a remote resource Y, it uses a temporary private key to sign a message that contains Y's identity and A's original delegation token, as shown in message (2). And this signed message becomes X's delegation token for Y. Note that this process may form a chain of delegation tokens with considerable less complexity than required in [13]. More details on computation complexity of [13] and our model will be shown in Section 5. A similar approach can be used for single sign-on by attaching an access token to the protocol messages. For instance, A could reuse the token that she received from S, $Sig_{K_S^*}[J_{Req}]$ (in Protocol 1) to authenticate herself to other resources provided S includes a list of resources that she is authorized to access within a certain valid period of time.

4 Supporting DKIG via TLS

As we have explained earlier, the use of params is similar to X.509 certificate in a PKI setting and thus it should be straightforward to integrate well-known protocols within DKIG. We present in Protocol 3, a modified version of Protocol 1 based on minor adjustments to the TLS handshake protocol [4]. We make use of the terminology from [4] with small changes where appropriate. For ease of exposition, we focus only on the interaction between A and X.

Protocol 3 Authenticated key establishment based on TLS handshake

(1) $A \rightarrow X$: `ClientHello.random =` r_A, t_A
(2) $X \rightarrow A$: `ServerHello.random =` r_X, t_X
`ServerParams =` params_X
`ParamsRequest`
`ServerHelloDone`
(3) $A \rightarrow X$: `ClientParams =` params_A
`ClientKeyExchange =` $E_{K_X^*}[K_{AX}]$
`AccessVerify =` $Sig_{K_S^*}[J_{Req}]$
`ClientFinished`
(4) $X \rightarrow A$: `ServerFinished`

As in the current TLS setting, Protocol 3 begins with the client (A) sending the server (X) a `ClientHello` message. The message contains a fresh random number and a timestamp as shown in step (1). X responds with a `ServerHello` message which contains an independent random number and a timestamp. X also forwards its `params` to A and makes a request for A's `params`. And then the `ServerHelloDone` message is sent to indicate the end of step (2).

In step (3), A first forwards her `params` to X. She then calculates a pre-master-secret as K_{AX} and forwards it encrypted under X's short-lived public key in the `ClientKeyExchange` message. We modify `CertificateVerify` in [4] to `AccessVerify` which contains the access token signed by S. A completes step (3) by sending `ClientFinished` that contains a verification value. This allows X to confirm that it has indeed received the previous handshake messages with the correct contents. The verification value can be easily established based on the specification in [4]. As with the `ClientFinished`, X should compute the exact verification value as A in `ServerFinished`. The shared secret key between A and X which will eventually be used for protecting application data over the connection is $master\_secret = \mathrm{PRF}(K_{AX}, \text{“}\texttt{mastersecret}\text{”}, r_A \| r_X)$, where PRF is a pseudo-random function.

Note that message (3) in Protocol 1 forms the input of `ClientKeyExchange` and `AccessVerify`. The handshake protocol will fail if the signature verification fails, or J_{Req} does not contain valid entries. If all succeed, X will compute its verification value and return it to A for confirmation.

5 Performance Trade-Off

We now summarise the computation complexity of single sign-on and delegation protocols for GSI when using the regular RSA-based PKI approach and DKIG using a conversion of a signature scheme in [3].

It is noticeable from Table 1 that both single sign-on and delegation services require the generation of a new RSA public/private key pair. This is computationally costly, particularly so if longer keys are used [13]. In addition, even

Table 1. Comparison between GSI and DKIG in terms of cryptographic operations in performing single sign-on and delegation services

Service	**Infra.**	**Operation**	
Single sign-on	GSI	1. Generation of new key pair 2. Signing of new proxy	
	DKIG	1. Signing of access token	
Delegation		A (delegator)	X (delegation target)
	GSI		1. Generation of new key pair 2. Signing of proxy request
		3. Verification of signed request 4. Signing of new proxy	
	DKIG	1. Signing of delegation token	

though signing and verification of signature in RSA scheme are computationally faster than the key generation, the delegation protocol needs at least two signing and one verification (for simplicity, we have not included verification of signature by the delegated target).

DKIG appears to offer significant advantages due to the use of fixed params, i.e. a user can create a new public key on-the-fly and requires no additional signing or verification operation. Even though the signing and verification algorithms in IBS are somewhat slower than RSA [1, 3], the total computation costs incurred for either the single sign-on or the delegation protocols are considerably lower than those encountered within an RSA-based PKI approach.

6 Conclusions

Current key management techniques used in PKI-based GRID solutions in particular for single sign-on and delegation protocols may well have high computation costs. We have presented a more lightweight and dynamic approach to providing a security infrastructure through what we term DKIG. In our DKIG setting which is based around the properties of IBC, each entity produces its own set of parameters that can be used by other entities as a permanent master-public-key mould. In this way, different temporary public keys can be tied to different jobs. Such short-lived public keys change dynamically from one job session to another, and thus improve key freshness and prevent key-replay attacks. We envisage such parameter sets to act as permanent credentials of the users just like their identities, and as such they are not expected to be renewed unless their master secret keys are compromised. More interestingly, the DKIG approach can be used to achieve single sign-on and rights delegation without the need for creating a proxy.

Acknowledgements

The authors would like to thank Sattam Al-Riyami, Chris Mitchell, Kenny Paterson, and anonymous referees for their feedback and comments.

References

1. P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology - Proceedings of CRYPTO 2002*, pages 354–368. Springer-Verlag LNCS 2442, 2002.
2. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology - Proceedings of CRYPTO 2001*, pages 213–229. Springer-Verlag LNCS 2139, 2001.
3. J.C. Cha and J.H. Cheon. An identity-based signature from Gap Diffie-Hellman groups. In Y.G. Desmedt, editor, *Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography - PKC 2003*, pages 18–30. Springer-Verlag LNCS 2567, 2003.

4. T. Dierks and C. Allen. The TLS protocol version 1.0. *The Internet Engineering Task Force (IETF)*, RFC 2246, January 1999.
5. I. Foster. The Grid: A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, February 2002.
6. I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational Grids. In *Proceedings of the 5th ACM Computer and Communications Security Conference*, pages 83–92. ACM Press, 1998.
7. H.W. Lim and M.J.B. Robshaw. On identity-based cryptography and GRID computing. In M. Bubak, G.D.v Albada, P.M.A. Sloot, and J.J. Dongarra, editors, *Proceedings of the 4th International Conference on Computational Science (ICCS 2004)*, pages 474–477. Springer-Verlag LNCS 3036, 2004.
8. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, pages 50–59. IEEE Computer Society Press, June 2002.
9. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. The community authorization service: Status and future. In *Proceedings of Computing in High Energy and Nuclear Physics (CHEP03), eConf*, March 2003.
10. A. Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology - Proceedings of CRYPTO '84*, pages 47–53. Springer-Verlag LNCS 196, 1985.
11. M.R. Thompson and K.R. Jackson. *Grid Resource Management: State of the Art and Future Trends*, chapter 5: security issues of Grid resource management, pages 53–69. Kluwer Academic, Boston, 2003.
12. S. Tuecke, V. Welch, D. Engert, L. Pearman, and M. Thompson. Internet X.509 public key infrastructure proxy certificate profile. *The Internet Engineering Task Force (IETF)*, RFC 3820, June 2004.
13. V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X.509 proxy certificates for dynamic delegation. In *Proceedings of the 3rd Annual PKI R&D Workshop*, 2004.
14. V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for Grid services. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12 2003)*, pages 48–61. IEEE Computer Society Press, June 2003.

Experiences of Applying Advanced Grid Authorisation Infrastructures

R.O. Sinnott[1], A.J. Stell[1], D.W. Chadwick[2], and O. Otenko[3]

[1] National e-Science Centre, University of Glasgow
{ros, ajstell}@dcs.gla.ac.uk
[2] Department of Computing Science, University of Kent
D.W.Chadwick@salford.ac.uk
[3] IS Security Research Centre, University of Salford
o.otenko@salford.ac.uk

Abstract. The widespread acceptance and uptake of Grid technology can only be achieved if it can be ensured that the security mechanisms needed to support Grid based collaborations are at least as strong as local security mechanisms. The predominant way in which security is currently addressed in the Grid community is through Public Key Infrastructures (PKI) to support *authentication*. Whilst PKIs address user identity issues, authentication does not provide fine grained control over what users are allowed to do on remote resources (*authorisation*). The Grid community have put forward numerous software proposals for authorisation infrastructures such as AKENTI [1], CAS [2], CARDEA [3], GSI [4], PERMIS [5,6,7] and VOMS [8,9]. It is clear that for the foreseeable future a collection of solutions will be the norm. To address this, the Global Grid Forum (GGF) have proposed a generic SAML based authorisation API which in principle should allow for fine grained control for *authorised* access to any Grid service. Experiences in applying and stress testing this API from a variety of different application domains are essential to give insight into the practical aspects of large scale usage of authorisation infrastructures. This paper presents experiences from the DTI funded BRIDGES project [10] and the JISC funded DyVOSE project [11] in using this API with Globus version 3.3 [12] and the PERMIS authorisation infrastructure.

1 Introduction

Today, collections of distributed individuals and institutions in science and industry are increasingly forming virtual organisations (VOs) to pool resources such as data sets, data archives, CPUs, or specialised equipment from astronomical radio-telescopes through to medical imaging scanners. Grid technology presents itself as one of the main ways in which such VOs can be established. With the open and collaborative nature of the Grid, ensuring that local security constraints are met and not weakened by Grid security solutions is paramount. PKIs represent the most common way in which security is addressed. Through PKIs, it is possible to validate the identity of a given user requesting access to a given resource. For example, with the Globus toolkit [12] solution, *gatekeepers* are used to ensure that signed requests

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 265 – 274, 2005.

are valid, i.e. from known collaborators. When this is so, i.e. the Distinguished Name (DN) of the requestor is in a locally stored and managed *gridmap* file, then the user is typically given access to the locally set up account as defined in the *gridmap* file.

There are several key limitations with this approach with regard to security however. Most importantly, the level of granularity of security is limited. There is no mention of what the user is allowed to do once they have gained access to the resource. Another issue with this approach is that it works on the assumption that user certificates are provided by an acknowledged certificate authority (CA). In the UK, a centrally managed CA at Rutherford Appleton Laboratories exists which (necessarily!) has strict procedures for how certificates are allocated. Users are expected to "prove" who they are in order to get a certificate, e.g. through presenting their passports to a trusted individual. This is a human intensive activity and one which is likely to have scalability issues once it is rolled out to the wider community, e.g. to industry and larger groups such as students taking Grid/e-Science courses. Having users personally take care of their private keys is another limitation of this approach.

In short, current experiences with PKIs [13, 14] as the mechanism for ensuring security on the Grid have not been too successful [15, 16]. Authorisation infrastructures offer extended and finer grained security control when accessing and using Grid resources. Numerous technological solutions have been put forward providing various levels of authorisation capabilities e.g. AKENTI [1], CAS [2], CARDEA [3], GSI [4], PERMIS [5,6,7] and VOMS [8,9]. Examples of how these compare to one another is described in [17, 18, 19]. It is too early to say if large scale use of attribute certificates (ACs) for user authorisation, based on infrastructures such as PERMIS, will be successful or not – more practical experience is required. In the current PERMIS infrastructure, static delegation of authority is supported, meaning that a central authority has to be contacted, and register local managers in its policy, before managers are entitled to assign privileges to subordinates. A better system is dynamic delegation of authority, where local managers do not need to be registered, but are given the privilege to delegate when they are first given privileges to use the system. Managers can then allocate privileges to staff and students as required, without having to contact the central authority first to get permission. Through this, a federated and scalable model of security authorisation can be realised that can be used for the dynamic establishment of VOs. VOs allow shared use of computational and data resources by collaborating institutions. Establishing a VO requires that efficient access control mechanisms to the shared resources by known individuals are in place. However, currently in the Grid community access control is usually done by comparing the authenticated name of an entity to a name in an Access Control List. This approach lacks scalability and manageability as discussed in [15]. Dynamic delegation of privileges offers a more realistic approach that could shape future Grid security, especially when it is rolled-out to the masses, e.g. Grid students and industry.

2 Authorisation Background

Authentication should be augmented with authorisation capabilities, which can be considered as what Grid users are allowed to do on a given Grid end-system. Thus

"what users are allowed to do" can be interpreted as the privileges that the users have been allocated on those end-systems. The X.509 standard [20] has standardised the certificates of a privilege management infrastructure (PMI). A PMI can be considered as being related to authorisation in much the same way as a PKI is related to authentication. Consequently, there are many similar concepts in PKIs and PMIs as discussed in detail in [6].

A key concept from PMI are attribute certificates (ACs) which, in much the same manner as public key certificates in PKI, maintain a strong binding between a user's name and one or more privilege attributes. The entity that digitally signs a public key certificate is called a Certification Authority (CA) whilst the entity that signs an AC is called an Attribute Authority (AA). The root of trust of a PKI is sometimes called the root CA – which in terms of the UK e-Science community is given by the Grid Support centre at RAL [21]. The root of trust of the PMI is called the Source of Authority (SOA). CAs may have subordinate CAs whom they trust and to which they delegate the powers of authentication and certification. Similarly, SOAs may delegate their power of authorisation to subordinate AAs. If a user needs to have their signing key revoked, a CA will issue a certificate revocation list. Similarly, if a user needs to have authorisation permissions revoked, an AA will issue an attribute certificate revocation list (ACRL). Typically, a given users' access rights are held as access control lists (ACLs) within each target resource. In an X.509 PMI, the access rights are held within the privilege attributes of ACs that are issued to users. A given privilege attribute within an AC will describe one or more of the user's access rights. A target resource will then read a user's AC to see if they are allowed to perform the action being requested.

The X.812 | ISO 10181-3 Access Control Framework standard [22] defines a generic framework to support authorisation. With this model *initiators* attempt to access a *target* in a remote domain. Two key components are put forward in [22] to support authorised access to the target: an Access control Enforcement Function (also known as a Policy Enforcement Point (PEP)) and an Access control Decision Function (also known as a Policy Decision Point (PDP)). The PEP ensures that all requests to access the target are authorised through checking with the PDP. The PDP's authorisation decision policy is often represented through collections of rules (policies), e.g. stored in a Lightweight Directory Access Protocol (LDAP) server. The different authorisation infrastructures associated with Grid technology have put forward their own mechanisms for realising PEPs and PDPs. Recently however, the GGF has put forward a generic API – the SAML AuthZ API - which in principle provides a generic PEP that can be associated with an arbitrary authorisation infrastructure [23]. The Grid specification is an enhanced profile of the OASIS Security Assertion Markup Language v1.1 [24]

2.1 GGF SAML AuthZ API

The OASIS SAML specification defines a number of elements for making assertions and queries regarding authentication, authorization decisions and attributes The OASIS SAML AuthZ specification defines a message exchange between a policy enforcement point (PEP) and a policy decision point (PDP) consisting of an *AuthorizationDecisionQuery* flowing from the PEP to the PDP, with an assertion

returned containing some number of *AuthorizationDecisionStatements*. The *AuthorizationDecisionQuery* itself consists of: *a Subject* element containing a *NameIdentifier* specifying the initiator identity; a *Resource* element specifying the resource to which the request to be authorized is being made, and one or more *Action* elements specifying the actions being requested on the resources. The GGF SAML profile specifies a *SimpleAuthorizationDecisionStatement* (essentially a granted/denied Boolean) and an *ExtendedAuthorizationDecisionQuery* that allows the PEP to specify whether the simple or full authorization decision is to be returned. In addition the GGF query supports both the pull and push modes of operation for the PDP to obtain attribute certificates, and has added a *SubjectAttributeReferenceAdvice* element to allow the PEP to inform the PDP where it may obtain the subject's attribute certificates from.

Through this SAML AuthZ API, a generic PEP can be achieved which can be associated with arbitrary (GT3.3) Grid services. Thus rather than developers having to explicitly engineer a PEP on a per application basis, the information contained within the deployment descriptor file (.wsdd) when the service is deployed within the container, is used. Authorisation checks on users attempting to invoke "***methods***" associated with this service are then made using the information in the .wsdd file and the contents of the LDAP repository (PDP) together with the DN of the user themselves. Note that this "method" authorisation basis extends current security mechanisms such as GSI which work on a per service/container basis. This generic solution can be applied to numerous infrastructures used to realise PDPs such as PERMIS.

2.2 PERMIS Background

The Privilege and Role Management Infrastructure Standards Validation (PERMIS) project [7] was an EC project that built an authorisation infrastructure to realise a scalable X.509 AC based PMI. Through PERMIS, an alternative and more scalable approach to centrally allocated X.509 public key certificates can be achieved through the issuance of locally allocated X.509 ACs.

The PERMIS software realises a Role Based Access Control (RBAC) authorisation infrastructure. It offers a standards-based Java API that allows developers of resource gateways (gatekeepers) to enquire if a particular access to a resource should be allowed. The PERMIS RBAC system uses XML based policies defining rules, specifying which access control decisions are to be made for given VO resources. These rules include definitions of: subjects that can be assigned roles; SOAs (local managers) trusted to assign roles to subjects; roles and their hierarchical relationships; what roles can be assigned to which subjects by which SOAs; target resources, and the actions that can be applied to them; which roles are allowed to perform which actions on which targets, and the conditions under which access can be granted to roles.

Roles are assigned to subjects by issuing them with X.509 Attribute Certificate(s). A graphical tool called the Privilege Allocator (PA) and a simpler version termed the Attribute Certificate Manager (ACM) have been developed to support this process. Once roles are assigned, and policies are developed, they are digitally signed by a manager and stored in one or more LDAP repositories.

To set up and administer PERMIS requires the use of a LDAP server to store the attribute certificates and reference the SOA root certificate. A local certificate authority (CA) is also required to be set up using OpenSSL – this designates the SOA and all user certificates created from this CA must have a Distinguished Name that matches the structure of the LDAP server. The DN of the user certificate is what is used to identify the client making the call on the grid service. Establishing local CAs matching the structures of the LDAP repository is not without issues which need to be resolved, e.g. in ensuring that locally generated certificates are recognised (trusted) by other remote CAs since there is no root of trust. From the user's perspective, once the administrator has set up the infrastructure, the PERMIS service is relatively easy to use. Unique identifiers are placed as parameters into the user's grid service deployment descriptor (.wsdd file). These are the Object Identification (OID) number of the policy in the repository, the URI of the LDAP server where the policies are held and the SOA associated with the policy being implemented. Once these parameters are input and the service is deployed, the user creates a proxy certificate with the user certificate created by the local CA to perform strong authentication. The client is run and the authorisation process allows or disallows the intended action.

3 Experiences of Authorisation

The GGF SAML AuthZ API offers, in principle, a generic way in which authorisation can be made. It is clear that direct experiences in applying/stress testing this mechanism are needed from a variety of different application domains. This has been undertaken within the BRIDGES project where the emphasis on security has been on life science data security, and the DyVOSE project where focus has been on education case studies looking at method level security.

3.1 Bridges Background

The BRIDGES project [10] is investigating the application of the Globus toolkit [12] to support HPC bioinformatics BLAST services using large HPC facilities; and the Open Grid Services Architecture – Data Access and Integration (OGSA-DAI) [26] and IBM's Information Integrator product [27] to deal with federation of distributed biomedical data for the Cardiovascular Functional Genomics (CFG) [25]. A key requirement of the scientist and hence focus of the BRIDGES work is security. Broadly speaking, the CFG scientific data can be classified dependent upon its security characteristics into three groups: public data (with no/minimal security, e.g. publicly curated genomic databases); shared data (belonging to the CFG scientists/consortia, e.g. shared research data sets); private data (belonging to given CFG sites and unavailable to anyone else, e.g. personal medical records).

A typical scenario that the infrastructure supports is: a user requests access to the CFG portal; the access request results in a SAML query being raised to ensure that this user is authorised to access the portal (by ensuring an appropriate role AC is available in the secure LDAP repository); if successful (the user is authorised), the portal is configured/personalised to display the services that are associated with that user; at this point, the user can invoke various services (they are entitled to use) – one

of these is a syntenic relation visualisation service (SyntenyVista); upon launching SyntenyVista (using WebStart technologies) the users can use data available in the repository (which itself provides an OGSA-DAI front end and exploits IBM Information Integator to integrate and where possible federate various remote public data resources); the user may then visually explore genomic data sets and potentially export these onto the high throughput computing resources ScotGrid for sequence similarity checking (BLAST) against other query sequences.

In the current implementation the usage of SyntenyVista offers direct visualisation of data sets available via the repository (from ensembl [28]). It is planned however that the user is restricted to seeing and visualising the data sets that they are entitled to see based upon their role within the CFG virtual organisation (VO), this applies also to the usage/invocation of GT3 based Blast services, i.e. that they will be restricted to those users *and* those data sets that meet appropriate security restrictions. For this purpose, the PERMIS Policy Editor tool has been used to develop appropriate policies based upon the specific roles in the projects and the capabilities to be associated with those roles.

3.1.1 Bridges Security Experiences

The emphasis on security in BRIDGES is upon data security. Work has investigated how best to map advanced Grid authorisation infrastructures such as PERMIS/SAML AuthZ with best practice in the database management systems (DBMS) world. DBMS have extensive experience in addressing security aspects, e.g. with how to ensure users access data that they are entitled to. The relation between how much authorisation should be done through Grid software and how much should be left to the DBMS is not always clear in the Grid community. Explorations of BRIDGES in this area are that the PERMIS (Grid) roles within the CFG VO (as extracted from the AC repository) are mapped against specifically established user views of data sets available via the DB2 data repository. However one issue that has been encountered with the SAML AuthZ profile is the lack of granularity in how users might invoke actions. For example, different actions may or may not be allowed depending upon the data that they wish to access and potentially change. The SAML AuthZ profile does not currently allow actions to be distinguished based upon the parameters that might be associated with them. As a result, the GT3 based BLAST service cannot be restricted to BLAST those data sets that are appropriate to the invoker. Instead, the SAML AuthZ specification supports either a SecureGrid BLAST service or a non-secure BLAST service. Thus when the portal is personalised per user/role, it is not possible to distinguish the usage of individual operations, e.g. to allow arbitrary invocations of actions where the data sets themselves might change.

Further, the identification of explicit targets and actions applicable to the data in the DB itself is not easily reconciled. A naïve approach would be for example to explicitly have read/write actions on contents of the database itself, e.g. read/write access to individual tables. The difficulty in this situation is that the DB is perpetually being modified (extended) as new data sets are added and changed. As a result, new policies would have to be defined with each DB change which impacts directly upon the scalability of the approach. In addition, attaching policies to individual data elements would face immediate scalability problems.

To address this issue, the project is investigating how the schemas defining the secure data structures themselves might be extended in a more scalable way to include security attribute information. Thus policies can be formulated to query data sets that do/do not have appropriate security attributes depending upon the policy in place. Through this mechanism, a generic approach to secure authorised access to DB contents can be achieved.

3.2 DyVOSE Background

The Dynamic Virtual Organisations in e-Science Education (DyVOSE) project [11] began in May 2004 and involves the Universities of Glasgow, Salford and in the second phase of the project, the University of Edinburgh. It was funded through the JISC Core Middleware programme.

One of the initial goals of DyVOSE is to explore scalability issues in the usage of advanced authorisation infrastructures such as PERMIS. To this extent, the PERMIS technology is being applied in the advanced MSc Grid Computing module at the University of Glasgow. It is worth noting that the first lecture had over 50 students.

Within the DyVOSE project the PERMIS tools such as the Policy Editor and Privilege Allocator have been used to create policies to authorise what the students are allowed to do as part of their programming assignment. To explore the authorisation infrastructure, the students have been asked to develop a GT3.3 service (*searchSortGridService*) which wraps a Condor based application (this service offers two methods to search (*searchMethod*) and sort (*sortMethod*) a large (5MB) text file (the complete works of Shakespeare)). The students themselves have been split into groups with the authorisation policy to ensure that method *sortMethod* can ***only*** be invoked by members of your student group and the lecturing staff, and that method *searchMethod* can be invoked by everyone.

Initially the students were asked to develop this policy themselves through the PERMIS Policy Editor. The usability of these tools is a key part in development of authorisation infrastructures. The output of the Policy Editor is an XML-based policy which identifies specific roles (*studentteam1, studentteam2* and *lecturer*), specific targets (*searchSortGridService*) and specific actions on that target (*searchMethod* and *sortMethod*). This XML policy is then input to the Privilege Allocator tool denotes specific users associated with that given rule (i.e. the students themselves); to digitally sign the policy and store it in the LDAP server.

3.2.1 DyVOSE Security Experiences

All of the students were able to successfully create the policy defined above using the PERMIS Policy Editor with minimal help from staff. It should be noted that the students were informed of various background information that they would need to create the policy including the Policy Domain to use ("O=University of Glasgow, C=GB"), the Source of Authority to use ("CN=Administrator, O=University of Glasgow, C=GB") and the Policy Object Identifier (1.0.0.1 for student group 1 and 1.0.0.2 for student group 2).

The students were requested to critically evaluate the PERMIS tools for this purpose, with these results being sent back to the PERMIS team for HCI improvements and minor bug fixes, e.g. problems in cross platform (Unix/Windows)

versions of the tool and functionality in the tool that has not yet been implemented (although the buttons/pull down menus exist). The student policies themselves were signed and stored as ACs within the LDAP server. At the time of writing the students are completing their assignment which is using these authorisation policies. The working solution demonstrating that these policies and the SAML AuthZ API are working has been produced however.

Establishing a working solution was not without issue however, e.g. one overhead is environment settings that must be configured before the PERMIS-GT3.3 solution can be used. The CLASSPATH environment variable, for instance, is sensitive to change: it must include most JAR files in the Globus installation library, but not include certain specific ones if an Ant build script is to be used to run the service client. Once these environment settings are identified, however, these can be incorporated into a script, which then only needs to be run once.

4 Conclusions and Future Plans

It is clear that detailed explorations are needed to assess the suitability of next generation Grid middleware. The work undertaken within the DyVOSE project has shown that the GGF SAML AuthZ API does provide a generic and useful mechanism through which fine grained authorisation can be achieved using GT3.3 and the PERMIS infrastructure. The BRIDGES project has shown the current limitations of this API which are being addressed by the GGF security authorisation working group through support for parameters in actions. Continued feedback on the PERMIS tools is an equally important activity. Students' experiences within the DyVOSE project are providing the PERMIS team with detailed feedback on the usability of these tools. These stem from needed functionality through to improvements to the HCI aspects of these tools.

The work in exploring the SAML AuthZ API has also identified issues with the Globus toolkit which have been fed back to the Globus team. Specifically, within the GT3.3 release, certain Globus source code was required to be commented out before PERMIS could run with it. Delays were also incurred due to the GT3.3 version compatible with PERMIS only being accessible via the CVS repository as opposed to the web site link. It is worth noting that it has been stated by the Globus team [30] that this SAML AuthZ API will be supported in future versions of the Globus software.

This work is addressing scalability issues of security infrastructures. A local central CA has issues with the overall manageability of PKIs, and does not address authorisation issues. A more realistic model would be to have a local CA infrastructure to issue certificates, e.g. to students as part of their matriculation. Within DyVOSE and BRIDGES a local certificate authority was established using OpenSSL [29]. Whilst relatively straightforward to achieve, there are issues in recognition of these certificates by other CAs within PKIs, such as the UK e-Science CA. Since no root of trust exists between these CAs, solutions might be based upon some form of bridging solutions [31]. However, given the limitations of PKIs a better solution would be to support dynamic establishment and recognition of trust to support authorisation. The second phase of the DyVOSE project will, through extensions to the PERMIS technologies, investigate how dynamic delegation of trust

can be achieved. In this situation, collections of distributed policies issued by various remote SOAs will be dynamically recognised (locally) and used as the basis for establishing the rules through which the dynamic VOs will be managed and enforced. This will benefit from the Shibboleth suite of protocols [33] for transport of policy information.

The explorations being undertaken in the BRIDGES and DyVOSE projects are providing valuable insight into the scalability and suitability of advanced authorisation infrastructures to establish VOs. These experiences are feeding in to numerous other areas. These include applications of Grid technology to establish VOs within the clinical science domain as part of the VOTES project [32], and as input to the UK e-Science Grid Engineering Task Force – specifically the action line associated with authentication, authorisation and accounting. Experiences in the application of the PERMIS infrastructure have also been presented to the UK e-Science Security Task Force as part of an on-going activity in establishing best practice and usage of Grid security software.

References

[1] Johnston, W., Mudumbai, S., Thompson, M. Authorization and Attribute Certificates for Widely Distributed Access Control, IEEE 7th Int. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (http://www-itg.lbl.gov/security/Akenti/)

[2] L Pearlman, et al., A Community Authorisation Service for Group Collaboration, in Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks. 2002.

[3] Lepro, R., Cardea: Dynamic Access Control in Distributed Systems, NASA Technical Report NAS-03-020, November 2003

[4] Globus Grid Security Infrastructure (GSI), http://www-unix.globus.org/toolkit/docs/3.2/gsi/index.html

[5] D.W.Chadwick, A. Otenko, E.Ball, Role-based Access Control with X.509 Attribute Certificates, IEEE Internet Computing, March-April 2003, pp. 62-69.

[6] D.W.Chadwick, A. Otenko, The PERMIS X.509 Role Based Privilege Management Infrastructure, Future Generation Computer Systems, 936 (2002) 1–13, December 2002. Elsevier Science BV.

[7] Privilege and Role Management Infrastructure Standards Validation project www.permis.org

[8] VOMS Architecture, European Datagrid Authorization Working group, 5 September 2002.

[9] Steven Newhouse, Virtual Organisation Management, The London E-Science centre, http://www.lesc.ic.ac.uk/projects/oscar-g.html

[10] BioMedical Research Informatics Delivered by Grid Enabled Services project (BRIDGES), www.nesc.ac.uk/hub/projects/bridges

[11] Dynamic Virtual Organisations in e-Science Education project (DyVOSE), www.nesc.ac.uk/hub/projects/dyvose

[12] Globus, http://www.globus.org

[13] R. Housley, T. Polk, Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructures, Wiley Computer Publishing, 2001.

[14] ITU-T Recommendation X.509 (2001) | ISO/IEC 9594-8: 2001, Information technology – Open Systems Interconnection – Public-Key and Attribute Certificate Frameworks.

[15] JISC Authentication, Authorisation and Accounting (AAA) Programme Technologies for Information Environment Security (TIES), http://www.edina.ac.uk/projects/ties/ties_23-9.pdf.
[16] Whitten, A., and Tygar, J. D. Why Johnny can't encrypt: a *u*sability evaluation of PGP 5.0. Paper presented at the 9th USENIX security symposium, Washington, 1999.
[17] D. Chadwick, O. Otenko, A Comparison of the Akenti and PERMIS Authorization Infrastructures, in Ensuring Security in IT Infrastructures, Proceedings of ITI First International Conference on Information and Communications Technology (ICICT 2003) Cairo University, Ed. Mahmoud T El-Hadidi, p5-26, 2003
[18] Conceptual AuthZ Framework and Classification (DOC) https://forge.gridforum.org/docman2/ViewCategory.php?group_id=55&category_id=458
[19] A.J. Stell, Grid Security: An Evaluation of Authorisation Infrastructures for Grid Computing, MSc Dissertation, University of Glasgow, 2004.
[20] ITU-T Rec. X.509 (2000) | ISO/IEC 9594-8. The Directory: Authentication Framework.
[21] UK e-Science Certification Authority, www.grid-support.ac.uk
[22] ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996, Security Frameworks for open systems: Access control framework
[23] V. Welch, F. Siebenlist, D. Chadwick, S. Meder, L. Pearlman, Use of SAML for OGSA Authorization, June 2004, https://forge.gridforum.org/projects/ogsa-authz
[24] OASIS. Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) v1.1,. 2 September 2003, http://www.oasis-open.org/committees/security/
[25] Cardiovascular Functional Genomics project, http://www.brc.dcs.gla.ac.uk/projects/cfg/
[26] Open Grid Service Architecture – Data Access and Integration project (OGSA-DAI), www.ogsadai.org.uk
[27] IBM Information Integrator, www.ibm.com
[28] EMBL-EBI European Bioinformatics Institute, http://www.ebi.ac.uk/ensembl/
[29] OpenSSL to create certificates, http://www.flatmtn.com/computer/Linux-SSLCertificates.html
[30] Von Welch/Jennifer Schopf personal communications.
[31] J. Jokl, J. Basney and M. Humphrey, Experiences using Bridge CAs for Grids, Proceedings of UK Workshop on Grid Security Practice - Oxford, July 2004
[32] Virtual Organisations for Trials and Epidemiological Studies project (VOTES), www.nesc.ac.uk/hub/projects/votes
[33] Shibboleth, http://shibboleth.internet2.edu/

Towards a Grid-wide Intrusion Detection System

Stuart Kenny and Brian Coghlan

Trinity College Dublin, Ireland
{stuart.kenny, coghlan}@cs.tcd.ie

Abstract. We describe SANTA-G (Grid-enabled System Area Networks Trace Analysis), an instrument monitoring framework that uses the R-GMA (Relational Grid Monitoring Architecture). We describe the CanonicalProducer, the component that allows for instrument monitoring, and how it would be used to construct the basis of a Grid-wide intrusion detection system.

1 The R-GMA

The Grid Monitoring Architecture (GMA) [2] of the Global Grid Forum (GGF), as shown in Figure 1, consists of three components: *Consumers*, *Producers* and a directory service, which in the R-GMA is referred to as a *Registry*.

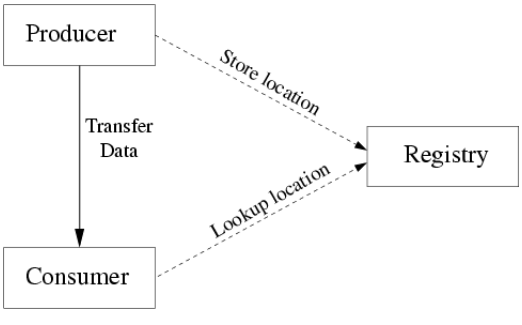


Fig. 1. Grid Monitoring Architecture

R-GMA is a relational implementation of the GMA developed within the European DataGrid (EDG), which harnesses the power and flexibility of the relational model. R-GMA creates the impression that you have one RDBMS per Virtual Organisation (VO). However it is important to appreciate that the system is a way of using the relational model in a Grid environment and *not* a general distributed RDBMS with guaranteed ACID properties. All the producers of information are quite independent. It is relational in the sense that Producers announce what they have to publish via an SQL CREATE TABLE

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 275–284, 2005.

statement and publish with an SQL INSERT and that Consumers use an SQL SELECT to collect the information they need. For a more formal description of R-GMA see [3]. The R-GMA makes use of the Tomcat Servlet container. Most of the R-GMA code is written in Java and is therefore highly portable. The only dependency on other EDG software components is in the security area.

There have so far been defined not just a single Producer but four different types: a DataBaseProducer, a StreamProducer, a LatestProducer and a CanonicalProducer. All appear to be Producers as seen by a Consumer, but they have different characteristics. The StreamProducer allows for information to be streamed continously to a Consumer. The LatestProducer only stores the most recent tuple of information for a given primary key, thus providing the latest-state information, whereas a DataBaseProducer stores the entire history of a stream of information.

2 The CanonicalProducer

If we have to deal with a large volume of data it may not be practical to convert it all to a tabular storage model. Moreover, it may be inefficient to transfer the data to a Producer servlet with SQL INSERT statements. It may be judged better to leave the data in its raw form at the location where it was created. The CanonicalProducer is able to cope with this by accepting SQL queries and using user-supplied code to return selected information in tabular form when required.

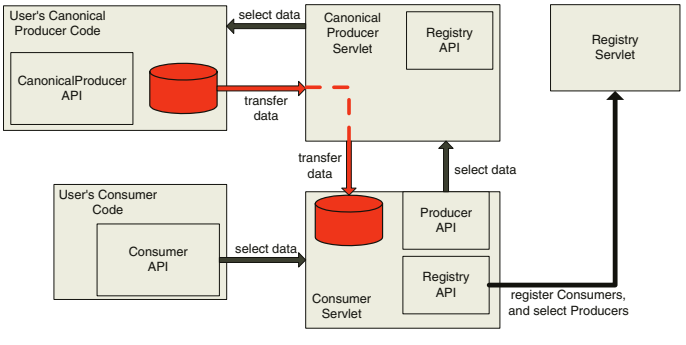


Fig. 2. CanonicalProducer Servlet Communication

In general the R-GMA producers are sub-classes of the Insertable class, the class that provides the insert method. The insert method is used by the producers to send data to the servlets as an SQL INSERT string. The CanonicalProducer is different however; it is a subclass of the Declarable class. This means that it inherits the methods for declaring tables, but not inserting data. The user's producer code is responsible for obtaining the data requested. Figure 2 shows the communication between the servlets for a CanonicalProducer. When the other

producer types publish data, the data is transferred to a local producer servlet via a SQL INSERT. The CanonicalProducer Servlet, however, is never sent raw data, which is instead retained local to the user's CanonicalProducer code.

Because the user must write the code to parse and execute the query, the CanonicalProducer can be used to carry out any type of query on any type of data source.

R-GMA is being further developed within the EU EGEE project [23]. A new Static query type has been added. The concept of the Canonical Producer has been extended as an On-Demand Producer that can encompass large databases as well as instruments, and which specifically supports the static query type.

3 SANTA-G

SANTA-G (Grid-enabled System Area Networks Trace Analysis) is a generic template for ad-hoc, non-invasive monitoring with external instruments, see Figure 3.

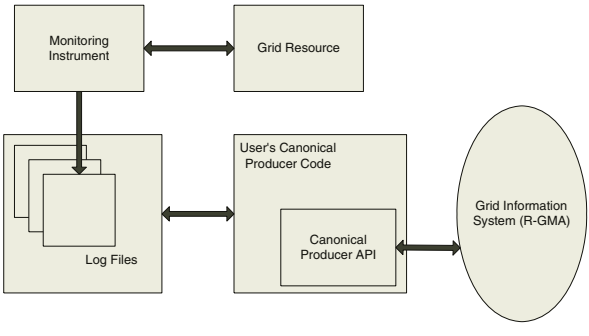


Fig. 3. SANTA-G monitoring framework

The template allows for the information captured by external instruments to be introduced into the Grid Information System. It is possible for these intruments to be anything, from fish sonars to PCR Analysers. The enabling technology for the template is the CanonicalProducer. The demonstrator of this concept, developed within the CrossGrid [17] project, is a network monitor that allows a user to access logs stored in *libpcap* (a packet capture library) format *through* the R-GMA. Examples of tools that generate logs in this format are Tcpdump [13], an open-source packet capture application, and SNORT.

SNORT [14] is an open-source network intrusion detection system. SNORT works by monitoring network packets received on the host's network interface card. Any packet found which matches a rule from a set of defined security rules is logged to a log file, and an alert is generated, which is also stored in a separate alert file.

4 Grid-wide Intrusion Detection

We now describe the use of the SANTA-G with SNORT as the basis for Grid-wide intrusion detection.

The SANTA-G NetTracer is composed of three components that allow for the monitoring data to be accessed through the R-GMA: a Sensor (which is installed on the node(s) to be monitored), a QueryEngine, and a Viewer GUI (see Figure 4). The Sensor monitors the log files created by the external sensor, for example SNORT, and notifies the QueryEngine when new log files are detected. SNORT logs alerts when suspect packets are detected. The Sensor monitors the alert file generated by SNORT and when a new alert is detected its details are sent to the QueryEngine, which records these events and publishes them to users through the R-GMA (by using the LatestProducer API). Users can then view these alerts, using the Viewer GUI. The full packet data of the packet that triggered the alert can also then be viewed by querying the packet log file also generated by SNORT.

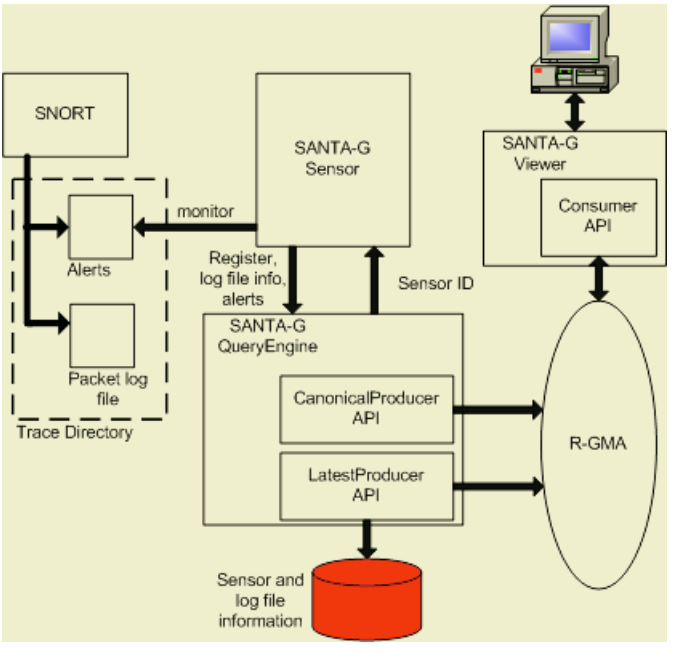


Fig. 4. SANTA-G NetTracer, SNORT monitoring

The QueryEngine provides the interface to the R-GMA by using the CanonicalProducer API. Data is viewed via the R-GMA by submitting an SQL SELECT statement, as if querying a relational database. Through the CanonicalProducer this query is forwarded to the QueryEngine, which then parses the

query, searches the appropriate log file to obtain the data required to satisfy the query, and returns the dataset to the GUI through the R-GMA.

The QueryEngine implements the required components of CanonicalProducer code. Figure 5 shows how the QueryEngine executes a SQL query received from the R-GMA (i.e. from the CanonicalProducerServlet).

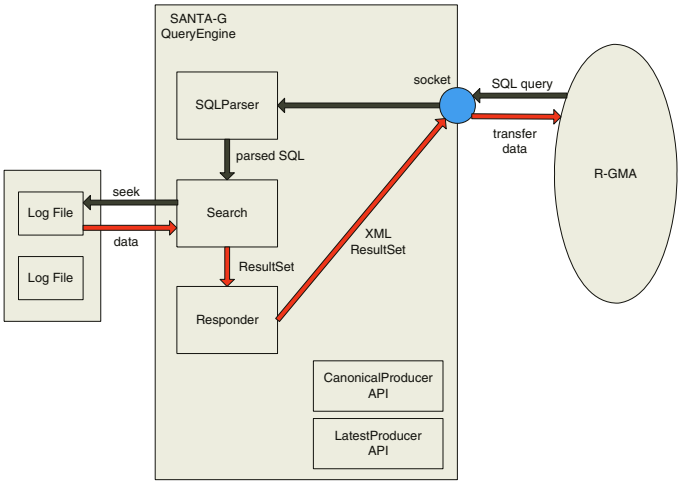


Fig. 5. SANTA-G QueryEngine Query Processing

The QueryEngine listens on a socket, waiting for connections from the Servlet. When a connection is made the SQL query is read from the socket and passed to an SQLParser class. The parser breaks the query into three separate lists; a select list that contains the fields to be read, a from list that contains the table the fields belong to, and a where list that contains the values used to filter the fields with. The Search class searches the log file for data that matches the WHERE predicates specified in the query, and extracts the required fields. The data that satisfies the query is accumulated into a ResultSet in XML format and returned to the Servlet over the socket connection. For example, the following query:

```
SELECT source_address, destination_address,
packet_type
FROM Ethernet
WHERE sensorId = 'some.machine.com:0'
AND fileId = 0
AND packetId < 100
```

would return the source address, destination address, and packet type fields of the Ethernet header for the first 100 packets in the log file assigned ID 0 and stored on 'some.machine.com'.

The information that can be queried is defined by the schema that the CanonicalProducer registers with the CanonicalProducer servlet. The schema for the SNORT alerts is shown in Table 1.

Table 1. SNORT alerts table schema

Field	Key	Description
siteId	PRI	Site ID
sensorId	PRI	Sensor ID
fileId	PRI	Log file ID
alert_timestamp		Timestamp of when the event was logged
alert_type		Type of event
sig_info		Alert signature information
message		Alert message
classification		Alert classification name
priority		Alert priority
source_ip		Source IP address
destination_ip		Destination IP address
source_port		TCP source port
destination_port		TCP destination port
data		Packet header data

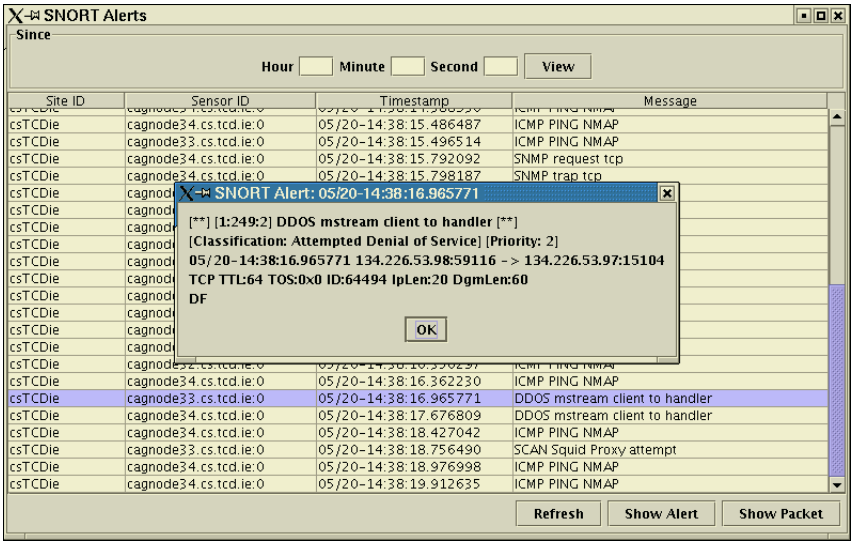


Fig. 6. SANTA-G Viewer, SNORT alerts panel

The SANTA-G Viewer provides a graphical user interface, which makes use of the R-GMA Consumer API, to allow users to graphically view network packets

in the log files, and also to build and submit SQL queries that will be carried out on the log files. Figure 6 shows the SNORT alerts panel of the Viewer, which allows a user to browse the alerts that have been published by the SNORT sensors to the R-GMA.

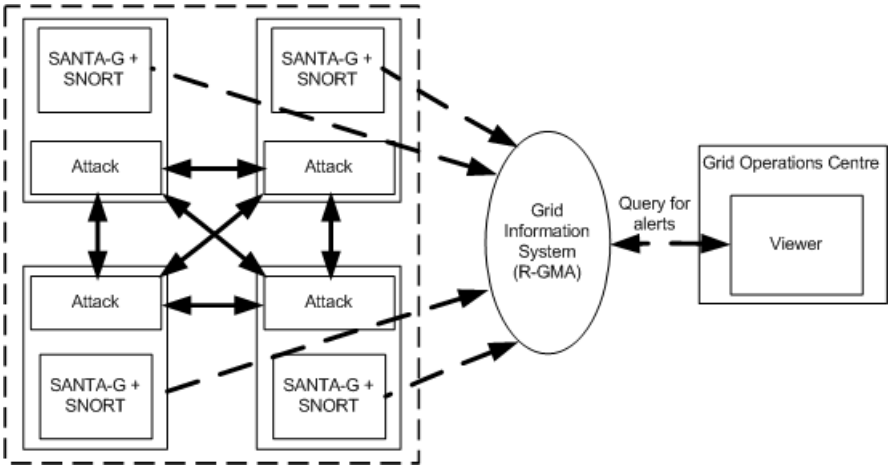


Fig. 7. SNORT monitoring example

Figure 7 shows how alerts can be generated, and published from a site, in order for them to be viewed at a Grid Operations Center. Here we show four worker nodes instrumented with the SNORT sensors, each attacking the other three. The alerts generated will be collected and published to the R-GMA via the QueryEngine. The Viewer (or any R-GMA Consumer) can then be used to query for and view the published alerts. If this example is expanded to include multiple sites then the alerts table published by the R-GMA becomes a Grid-wide intrusion log.

5 Future Work

It is intended to use the SNORT functionality of the SANTA-G network monitoring tool developed within Grid-Ireland to construct a 'Grid-wide intrusion detection system'. It is envisaged that each site within a Grid will run a set of SNORT sensors publishing alerts to the R-GMA. A high-level incident detection, tracking and response platform will be created by using custom coded Consumers to filter and analyse the alerts published in order to detect patterns that would signify an attempted distributed attack on the Grid infrastructure. This will be constructed with two components (as shown in Figure 8):

1. a R-GMA Archiver that will query for the alerts detected by the SNORT sensors and save the results in a MySQL database, that will then represent the Grid-wide intrusion log, and

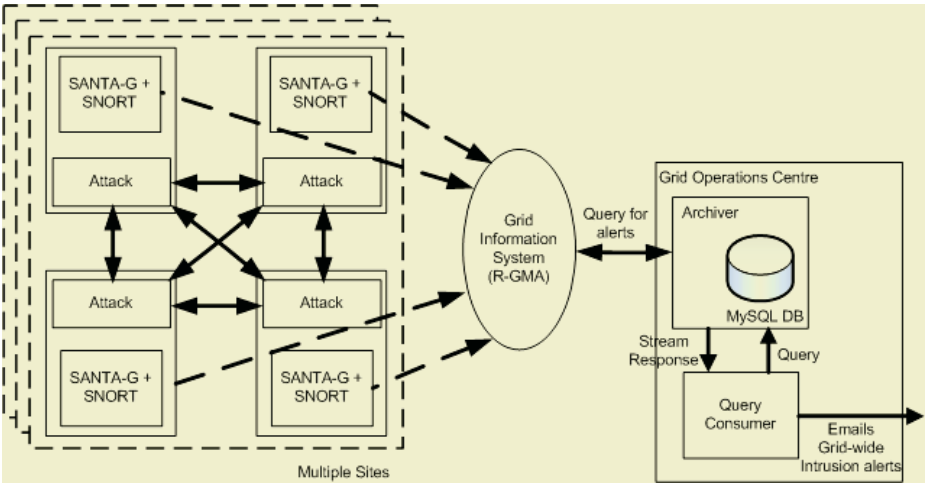


Fig. 8. Grid-wide Intrusion Detection System

2. one or more R-GMA Consumers that will issue stream queries to the Archiver, and receive back a stream of responses that they will convert to email alerts.

It is also intended to complement the SNORT data with information from other security components. Work is currently underway creating new sensor types and query engines for use with such tools as Tripwire [15] and AIDE (Advanced Intrusion Detection Environment) [16].

6 Conclusion

The SANTA-G network monitoring tool developed within the CrossGrid project demonstrates the CanonicalProducer concept by allowing log files to be accessed *through* the R-GMA. Since the logs are published to the R-GMA grid information system this can provide the basis for a Grid-wide intrusion detection system. SANTA-G NetTracer has been extended to publish logs generated by the SNORT network intrusion detection system. We intend to use this functionality, along with further extensions that incorporate information from other security components, to construct a system that will allow for Grid-wide intrusion detection. By using other R-GMA components, such as an Archiver, alerts published from multiple sites can be aggregated to form a Grid-wide intrusion log. Custom coded Consumers can then query this log for specific alert patterns, triggering their own alerts if a pattern is detected, and thereby generating Grid-wide intrusion alerts.

References

1. Andrew Cooke, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Ari Datta, Roney Cordenonsi, Rob Byrom, Laurence Field, Steve Hicks, Manish Soni, Antony Wilson, Xiaomei Zhu, Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Norbert Podhorszki, Brian Coghlan, Stuart Kenny David O'Callaghan, John Ryan. RGMA: First Results After Deployment CHEP03, La Jolla, California, March 24-28, 2003.
2. B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. *A Grid monitoring architecture.* Global Grid Forum Performance Working Group, March 2000. Revised January 2002.
3. Andy Cooke, Alasdair J G Gray, Lisha Ma, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Rob Byrom, Laurence Field, Steve Hicks, Jason Leake, Manish Soni, Antony Wilson, Roney Cordenonsi, Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Norbert Podhorszki, Brian Coghlan Stuart Kenny, David O'Callaghan. *R-GMA: An Information Integration System for Grid Monitoring* Proceedings of the Tenth International Conference on Cooperative Information Systems, 2003.
4. Andrew Cooke, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Ari Datta, Roney Cordenonsi, Rob Byrom, Laurence Field, Steve Hicks, Manish Soni, Antony Wilson, Xiaomei Zhu, Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Norbert Podhorszki, Brian Coghlan, Stuart Kenny, Oliver Lyttleton, David O'Callaghan, John Ryan. *Information and Monitoring Services Within a Grid* Proceedings of the Eleventh International Conference on Cooperative Information Systems, 2004.
5. Brian Coghlan, Abdeslem Djaoui, Steve Fisher, James Magowan, Manfred Oevers. *Time, Information Services and the Grid* 31st May 2001.
6. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, P. Vanderbilt. *Grid Service Specification* http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-04_2002-10-04.pdf, 2003.
7. The DataGrid Project. http://www.eu-datagrid.org
8. DataGrid WP3. *DataGrid Information and Monitoring Final Evaluation Report* https://edms.cern.ch/document/410810/4/DataGrid-03-D3.6-410810-4-0.pdf
9. Brian Coghlan, Stuart Kenny. *SANTA-G Software Design Document*
10. Brian Coghlan, Stuart Kenny. *SANTA-G First prototype Description* http://www-eu-crossgrid.org/Deliverables/M12pdf/CG3.3.2-TCD-D3.3-v1.1-SANTAG.pdf
11. CrossGrid WP3. *Deliverable D3.5, Report on the Results of the 2nd and 3rd Prototype* http://www-eu-crossgrid.org/Deliverables/M24pdf/CG3.0-D3.5-v1.2-PSNC010-Proto2Status.pdf
12. Brian Coghlan, Andrew Cooke, Roney Cordenonsi, Linda Cornwall, Ari Datta, Abdeslem Djaoui, Laurence Field, Steve Fisher, Steve Hicks, Stuart Kenny, James Magowan, Werner Nutt, David O'Callaghan, Manfred Oevers, Norbert Podhorszki, John Ryan, Manish Soni, Paul Taylor, Antony Wilson Xiaomei Zhu. *The CanonicalProducer: an instrument monitoring component of the Relational Grid Monitoring Architecture* Proceedings of the 3rd International Symposium on Parallel and Distributed Computing, July 2004.
13. Tcpdump. http://www.tcpdump.org
14. SNORT. http://www.snort.org
15. Tripwire. http://www.tripwire.org
16. AIDE. http://sourceforge.net/projects/aide

17. The CrossGrid Project http://www.eu-crossgrid.org
18. DataGrid WP3 Information and Monitoring Services http://hepunx.rl.ac.uk/edg/wp3/
19. Global grid forum. http://www.ggf.org
20. I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 2: Computational Grids, pages 15–51. Morgan Kaufmann, 1999.
21. I. Foster, C. Kesselman, and S. Tuecke. *The anatomy of the Grid: Enabling scalable virtual organization.* The International Journal of High Performance Computing Applications, 15(3):200–222, 2001.
22. Globus Toolkit. http://www.globus.org
23. Enabling Grids for E-science in Europe http://egee-intranet.web.cern.ch/egee-intranet/gateway.html

International Grid CA Interworking, Peer Review and Policy Management Through the European DataGrid Certification Authority Coordination Group

J. Astalos[13], R. Cecchini[14], B. Coghlan[6], R. Cowles[20], U. Epting[11], T. Genovese[8], J. Gomes[15], D. Groep[18], M. Gug[9], A. Hanushevsky[20], M. Helm[8], J. Jensen[3], C. Kanellopoulos[1], D. Kelsey[3,⋆], R. Marco[12], I. Neilson[9], S. Nicoud[5], D. O'Callaghan[6], D. Quesnel[2], I. Schaeffner[11], L. Shamardin[16], D. Skow[10], M. Sova[4], A. Wäänänen[17], P. Wolniewicz[19], and W. Xing[7]

[1] Aristotle University of Thessaloniki, Greece
[2] Canarie, Canada
[3] Rutherford Appleton Laboratory, UK
[4] CESNET, Czech Republic
[5] CNRS/UREC CPPM, France
[6] Trinity College Dublin, Ireland
[7] University of Cyprus, Cyprus
[8] ESnet/LBNL, USA
[9] European Organization for Nuclear Research (CERN), Switzerland
[10] Fermi National Accelerator Laboratory, USA
[11] Forschungszentrum Karlsruhe, Germany
[12] Instituto de Física de Cantabria (CSIC-UC), Spain
[13] Slovak Academy of Sciences, Slovakia
[14] INFN, Italy
[15] Laboratório de Instrumentação e Física Experimental de Partículas, Portugal
[16] Moscow State University, Russia
[17] Niels Bohr Institute, Denmark
[18] NIKHEF, Netherlands
[19] Poznań Supercomputing and Networking Center, Poland
[20] Stanford Linear Accelerator Center, USA

Abstract. The Certification Authority Coordination Group in the European DataGrid project has created a large-scale Public Key Infrastructure and the policies and procedures to operate it successfully. The infrastructure demonstrates interoperability of multiple certification authorities (CAs) in a novel system of peer-assessment of the roots of trust. Crucial to the assessment is the definition of minimum requirements that all CAs must meet in order to be accepted. The evaluation is aided by software-generated trust matrices. Related work building on this infrastructure is described. The group's policies and experience now form the basis of the new European Policy Management Authority for Grid Authentication in e-Science.

⋆ Corresponding author: D.P.Kelsey@rl.ac.uk

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 285–295, 2005.

1 Introduction

This paper describes the creation and operation of a Public Key Infrastructure (PKI) for grid authentication used by several international grids. The European DataGrid (EDG) project[1], which began in January 2001, was the first European project to establish a wide-scale grid. During the three years of EDG, the authentication requirements of this and other grid projects led to the inclusion of 21 Certification Authorities (CAs) in the PKI. These CAs provide authentication services for people and grid services in the majority of the EU member states and also in Canada, Russia, Taiwan and the USA.

EDG was the first grid project to involve more than a small number of nations, each with their own administrative and security domains. Initially this was not perceived as an issue, but project members soon realised that the resource owners required a more structured approach to security. The Certification Authority Coordination Group (CACG) was established at the beginning of the project to define a common authentication infrastructure trusted by all relying parties that were part of the EDG project. EDG's sister projects, such as DataTAG[2] and CrossGrid[3] have adopted the EDG security model. GridLab[4] recognises the CACG member CAs. The LCG[5] and EGEE[6] projects also take the EDG approach to authentication.

EDG security activities fell into three categories: authentication, authorization and coordination. EDG decided to keep authentication and authorization separate (due to the more dynamic nature of authorization) while recognizing that authentication often includes some implicit authorization. Authentication was based on the Globus Grid Security Infrastructure (GSI)[7]. The Security Coordination Group (SCG) have documented the EDG authorization developments [8, 9, 10].

2 DataGrid Authentication

The EDG SCG collected and documented the security requirements of the project [11]. These included 17 requirements for authentication of which three important items were: for a user to authenticate just once per session; interoperable authentication between many grids and applications; and the ability of authentication to be revoked in the event of loss or compromise of an identity credential. The requirements led to the use of Globus GSI. This uses a Public Key Infrastructure (PKI) with X.509 certificates[12]. Identity is checked by a Registration Authority (RA) and certified by a Certification Authority (CA). Users, hosts and services perform mutual authentication. Delegation with short-lifetime proxy credentials achieves the important goal of single sign-on[13]. A grid mapfile maps a certificate's distinguished name (DN) to a local account and authorization is enforced by the local security mechanisms.

The CA Coordination Group had the task of creating a PKI, which was unique in its successful coordinated use of the technology with a large number of independently operated CAs. The infrastructure was for GSI authentication only:

it specifically did not support long-term encryption or digital signatures. A single certification authority for the whole project was not thought to be sufficient due to concerns about a single point of failure or attack. It was also important to have robust relationships between each CA and its associated RAs. To meet these requirements, an appropriate scale was one CA for each country, large region or international organization. A single hierarchy would have excluded some pre-existing CAs, reduced the ability of CAs to meet local needs, and was not convenient to support with the Globus software. For these reasons a coordinated group of peer CAs was the most suitable choice. The EDG project did not have any resources allocated to run such a PKI, so efforts were drawn from participating national projects and organizations.

2.1 Globus Grid Security Infrastructure Features

In Globus GSI the end-entity certificate is used to sign a 'proxy' certificate. In the validation of the proxy certificate, the end-entity basicConstraints (which state the that certificate is not a CA certificate) are deliberately ignored, a violation of the normal validation procedures. GSI proxy certificates are now an IETF standard described in RFC 3820 [14].

X.509 CRLs[15] have a nextUpdate field that conveys a hint when a new CRL can be obtained. In GSI, this field is interpreted strictly as an expiration date: if the CRL for a particular CA is present but outdated, end-entity certificates signed by this CA will not be accepted by the software.

2.2 Status of DataGrid PKI

At the end of the EDG project there were 21 approved national certification authorities. CNRS, France ran a 'catch-all' CA, for those without a national CA, with appropriate RA mechanisms. In Table 1, 'Total Issued' certificates include those for users, hosts and services and also includes certificates which have since expired or been revoked. In the 'Currently Valid' column is the current number of active certificates. The data for this table were collected in April 2004. Note: The CERN CA serves the CERN community. The FNAL Root CA is accredited but only issues CA certificates.

The CAs in the PKI each provide an equivalent service but with different resources. Many CAs use OpenSSL[16] . Others use Globus Simple CA[17] and various versions of OpenCA[18]. The DOEGrids CA uses Sun ONE[19] Certificate Server.

The relying parties, i.e. users, services and resources, of the PKI must be able to download and install the CA certificates, namespace signing policies, and CRLs of each trusted CA in a secure and robust way. The Certification Authority Repository[1] provides this information for grid administrators. CA information is distributed in RPM (RedHat Package Manager) format for the

[1] Certification Authority Repository: http://marianne.in2p3.fr/datagrid/ca/

Table 1. certification authority statistics

CA	Country	Total Issued	Currently Valid
ArmeSFo	Armenia	1	1
ASCCG	Taiwan	80	68
CERN	CERN	640	321
CESNET	Czech Republic	365	211
CNRS	France & Catch-all	1400	392
CyGrid	Cyprus	18	14
DataGrid-ES	Spain	408	191
DOEGrids	USA	2807	1572
GridCanada	Canada	570	467
Grid-Ireland	Ireland	170	111
GridKA	Germany	364	225
HellasGrid	Greece	49	33
INFN	Italy	1956	1158
LIP	Portugal	61	43
NIKHEF	Netherlands	321	124
NorduGrid	Nordic Countries	579	316
PolishGrid	Poland	266	207
Russian DataGrid	Russia	230	99
SlovakGrid	Slovakia	26	18
UK e-Science CA	UK	1856	1297
Total		12167	6868

EDG testbed. Scripts have been written to update CRLs periodically, as they are not fetched automatically by the Globus software.

3 Minimum Requirements for Grid Certification Authorities

One of the major activities of the CACG has been the production and maintenance of a set of minimum requirements and best practices for an "acceptable and trustworthy" CA as defined by the relying parties of EDG and related grid projects, taking into account the level of risk associated with the assets to be protected. These requirements have evolved during the project largely as a result of the numerous difficulties that arise when interoperating between different linguistic, administrative, networking and security domains. This section is based on the Minimum Requirements document of the European Policy Management Authority for Grid Authentication in e-Science (EUGridPMA), which is publicly available from http://www.eugridpma.org/.

In this section, the key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' are to be interpreted as described in RFC 2119. Text in *italics* provides discussion and clarification of the requirements.

Due to certain idiosyncrasies of the grid middleware, the PKI structure SHOULD NOT follow the conventional hierarchical model: there SHOULD be one certification authority (CA) per country, large region or international organization each with an associated network of registration authorities (RA). The RAs handle the tasks of validating the identity of the end entities and authenticating their requests, which will then be forwarded to the CA. The CA will handle the tasks of issuing CRLs; signing certificates and CRLs; and revoking certificates when necessary.

Requirements of the Certification Authority:

Computer Security Controls: The CA computer, where certificates are signed, SHOULD be a **dedicated machine**, running only services needed for CA operations. It MUST be located in a secure environment where access is limited to specific trained personnel and MUST be kept disconnected from any kind of network. If the CA computer is equipped with at least a FIPS 140-1 level 3 Hardware Security Module or equivalent it MAY be connected to a highly protected/monitored network. The security controls MUST be documented and the documentation made available to the PMA.

CA Namespace: Each CA MUST sign only a well defined namespace that does not clash with any other CA.

Policy Document & Identification: Every CA MUST have a Certification Policy and Certification Practice Statement (CP/CPS) and assign it an OID (object identifier). Whenever there is a change in the CP/CPS the OID of the document MUST change and changes MUST be approved by the PMA before signing any certs under the new CP/CPS. All the CP/CPSs under which valid certs are issued MUST be available on the web. *We currently recommend the RFC 2527 template for the CP/CPS document.*

CA Key: The CA Key MUST have a minimum length of 2048 bits and, for CAs that issue end-entity certificates, the lifetime MUST be no longer than 5 years and no less than twice the maximum life time of an end-entity certificate. The private key of the CA MUST be protected with a pass phrase of at least 15 characters and known **only** by specific personnel of the certification authority. A copy of the encrypted private key MUST be kept on an offline medium in a secure place. The pass phrase of the encrypted private key MUST also be kept on an offline medium in a secure place, separate from the key.

CA Certificate: The CA certificate MUST have the extensions keyUsage and basicConstraints marked as critical.

CRLs: The maximum CRL lifetime MUST be at most 30 days and the CA MUST issue a new CRL at least 7 days before expiration and immediately after a revocation. The CRLs MUST be published in a repository accessible via the World Wide Web, as soon as issued. *We recommend that all relying parties update their local copies of CRLs at least once per day.*

Records Archival: The CA MUST record and archive all requests for certificates, along with all the issued certificates; all the requests for revocation; all the issued CRLs; and the login/logout/reboot records of the issuing machine.

Key Changeover: The CA's private signing key MUST be changed periodically; from that time on only the new key will be used for certificate signing purposes. The overlap of the old and new key MUST be at least the longest time an end-entity cert can be valid. The older but still valid certificate MUST be available to verify old signatures, and the private key to sign CRLs, until all the certificates signed using the associated private key have expired.

Repository: The repository MUST be run on a best-effort basis, with an intended availability of 24×7.

Compliance Audits: Each CA MUST accept being audited by other CAs to verify its compliance with the rules and procedures specified in its CP/CPS document.

Operational Audits: The CA MUST perform operational audits of the CA and RA staff at least once per year.

Requirements of the Registration Authority:

Entity Identification: In order for an RA to validate the identity of a person, the subject MUST contact the RA personally and present photographic identification and/or valid official documents showing that the subject is an acceptable end entity as defined in the CA's CP/CPS. In case of host or service certificate requests, the request MUST be delivered to the RA by the person in charge of the specific entities using a secure method.

Name Uniqueness: The subject name listed in a certificate MUST be unambiguous and unique for all certificates issued by the CA.

Records and Archival: The RAs MUST record and archive all requests and confirmations.

Communication with CA: The RA MUST communicate with the CA with secure methods that are clearly defined in the CP/CPS. *e.g. signed emails, voice conversations with a known person, SSL protected mutually authenticated private web pages .*

The end-entity (EE) keys MUST be at least 1024 bits long and MUST NOT be generated by the CA or the RA. The EE certificates MUST have a maximum lifetime of 1 year and MUST NOT be shared among end entities. The EE certificate MUST contain information to identify which CP/CPS was used to issue the certificate (e.g. OID or date). The extensions basicConstraints and keyUsage MUST be marked as critical and the basicConstraints MUST be set to "CA: False". The CA SHOULD make a reasonable effort to make sure that end-entities understand the importance of protecting their private key, with a pass phrase of at least 12 characters.

4 Trust Evaluation

To establish trust, each CA is required to demonstrate to the group that the setup and policies are secure. This is usually done in person at a meeting of the CACG where detailed questions about the CP/CPS, the practices, the RA structure, etc. are answered. After satisfying this peer review a CA will be 'accredited'. Each relying party (RP) wants to evaluate all the CAs, either that they meet the RP's standard, or that they meet an agreed common standard. The CACG peer review establishes this common standard. This requires inspection of each CA's CP/CPS by a volunteer subset of the other CAs. Third-party audits have been considered but would be time-consuming and expensive and none have yet been done. Evaluation of trust is a continuous and long-term process and experience has shown that personal contacts are fundamental. The Global Grid Forum (GGF)[20] has established several working groups to establish policies and procedures in this area.

The assessment process is manual, and CA managers want to make it more automatic. Software is being developed to aid this process based on evaluation of a CA *Feature Matrix*. CP/CPS documents are encoded in a report file and the Feature Matrix displays the features. The CA report file uses a basic contextual language involving key-value pairs, e.g. name = 'CERN CA'. The language is designed to enable later extension to allow formal analysis, but is presently very simple. Features can be evaluated relative to *rulesets*. A *default ruleset* has been defined for EDG, based on the CACG minimum requirements. This allows the construction of a CA *Acceptance Matrix*.[2] The GGF concept of assurance levels is accommodated to allow rulesets to be defined for each level[21]. Each Virtual Organization (VO) can also define their own rules that override and extend the default ruleset. The *Ruleset Inclusion Principle* extends from the general to the specific. It can be extended to CAs, sites, hosts, users and even specific services simply by defining the appropriate ruleset. Thus a typical chain might be: default ruleset → VO ruleset → host ruleset. It is not necessary for a subject to have all possible rulesets in their possession, only those rulesets that they are interested in. Further evaluations with example user, host and service certificates, and samples of issued certificates are possible and this is the current focus. There are other complementary approaches: for example, evaluating an XML encoding of a CP/CPS[22].

5 Related Work

5.1 Certificate Request Applets

Java applets have been developed to be used for certificate requests. An applet generates the keys and associated request and submits them to the CA. Another applet is used to download the certificate and match it with the corresponding

[2] CA Trust Matrices: http://www.cs.tcd.ie/coghlan/cps-matrix/cps-matrix.cgi

private key. Once the certificate and key have been matched, they are exported in PKCS #12 format which can be imported into a browser.

The applets must be signed, since they read and write files on the user's disk, and so that the user trusts that the applets were issued by the CA. An advantage of using applets is that the CA can perform some basic validation when the user applies for the certificate, rather than rejecting invalid requests at a later stage. The applet method allows the CA to check the strength of the user's passphrase without ever seeing the passphrase or the private key. This is a great advantage over the 'normal' method where the user must be trusted to generate a sufficiently strong passphrase.

5.2 Compromised and Exposed Private Keys

The CACG has explored the issues related to the compromise and exposure of private keys. Compromised keys should be revoked, but the definition of a 'compromise' of credential confidentiality is unclear. It is *a priori* impossible to prove confidentiality to a third party, so we must rely on best professional judgement. This necessarily means cases will have to be evaluated individually. The following cases provide a working definition for 'compromise' and 'exposure' of private keys.

If a private key can be shown to be in the possession of someone other than the user then it is considered 'compromised'. When an attacker has had access to the user's unencrypted private key, it will be considered a 'compromise' unless forensic analysis can rule out access to the key. Compromised keys must be revoked.

If an encrypted private key is available to someone other than the user then it is considered 'exposed'. An encrypted private key is vulnerable to offline attack, protected only by the user-chosen passphrase. When an attacker has had access to the user's encrypted private key and the attacker demonstrates sufficient skill and knowledge of PKI, it will be considered a 'compromise' unless forensic analysis can rule out access to the key. Exposed keys should be reported to the appropriate CA who will alert the user to the exposure. Keys visible in the course of system administration will not normally be considered exposed.

5.3 Online Certificate Services

Traditionally, grid certification authorities have been operated offline. This reduces the risk of compromise of the CA signing key. Online certificate services are those which store private keys, and generate or sign certificates on a network-connected system. LCG[5] is using a KCA and ESnet is proposing a minimum requirements profile for online services.

5.3.1 Kerberized Certification Authority

The Kerberized Certification Authority (KCA) provides a automated mechanism for an organization with an existing Kerberos infrastructure to generate X.509 credentials for use in PKI-based authentication systems. The KCA software is distributed by the NSF Middleware Initiative (NMI)[23].

The KCA consists of a secure server which communicates with a client to generate PKI credentials. The KCA service is attractive for sites operating a Kerberos-based authentication infrastructure. The user is not issued a long term private key and proxy maintenance uses the existing Kerberos infrastructure. The administrative overhead and possibility of error or deliberate attack on another RA is removed. Since the KCA issues only short-lived certificates, there is no need to distribute CRLs. Compared to a well run offline service the danger of signing key compromise is increased. In the context of long-running jobs in the grid the problem arises of how to renew a proxy certificate derived from a user's Kerberos token which is typically valid for about one day.

5.3.2 Virtual Smart Cards

The SLAC Virtual Smart Card system[24], provides an online credential store analogous to a physical smart card. As users cannot be trusted to keep private keys secure they should not be given the private key. VSC can provide stronger security guarantees with a central restricted-access server than individual untrustworthy users, and it allows users to generate proxy certificates from anywhere that has access to the VSC server. The disadvantages are that the private keys are concentrated in one place, therefore giving a single point of failure, and the authentication for the whole system is only as strong as the authentication with the VSC server, so this must be of high quality, e.g. a well-administered Kerberos setup.

6 Summary

During the last three years the Certification Authorities Coordination Group has successfully built a large-scale Public Key Infrastructure which is now in global production use. This infrastructure allows users and services to have just one identity credential which is accepted and trusted by a growing number of VOs and grid projects.

The evolution of the best practices, minimum requirements and the associated establishment of inter-domain trust via peer review on behalf of the various relying parties, has taken time and involved many debates during the meetings of the group. The tools developed for trust evaluation and the various technical challenges of grid authentication have enabled the group to avoid having to spend all of its time concentrating on policies and procedures. As described in the paper, future work building on this infrastructure has already started. The expected growth of online certificate services and repositories, together with online certificate status checking, is likely to play a significant role in future authentication services.

The policies of the Certification Authority Coordination Group worked extremely well for EDG. With the input from other grid projects it has become a large group and now forms the basis of the new European Policy Management Authority for Grid Authentication in e-Science (EUGridPMA)[25]. This

new body, which is initially coordinating authentication services for EGEE[6], DEISA[26], LCG[5] and SEEGRID[27], is associated with the global Grid Policy Management Authority[28] initiative, started in 2002 to coordinate the PMAs. The policies, procedures and technical solutions developed by CACG and described in this paper, are being taken forward by the EUGridPMA with the aim of turning this into an even more pervasive general infrastructure for authentication for e-Science.

The authors wish to acknowledge the EU and many national funding bodies, institutes and projects that allowed their staff to participate in the activities of the Certification Authority Coordination Group. We thank all our colleagues in each of the grid projects, particularly European DataGrid, for providing very valuable comments and feedback on the authentication infrastructure during the project.

References

1. European DataGrid. (2004) http://www.edg.org/.
2. DataTAG. (2004) http://datatag.web.cern.ch/.
3. CrossGrid. (2004) http://www.crossgrid.org/.
4. GridLab. (2004) http://gridlab.org/.
5. LHC Computing Grid. (2004) http://lcg.web.cern.ch/.
6. Enabling Grids for E-science in Europe. (2004) http://www.eu-egee.org/.
7. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: ACM Conference on Computers and Security. ACM Press (1998) 83–91
8. DataGrid Security Coordination Group: Security Design. (2003) https://edms.cern.ch/document/344562.
9. DataGrid Security Coordination Group: Final Security Report. (2004) https://edms.cern.ch/document/414762.
10. Cornwall, L.A. *et al.*: Security in multi-domain grid environments. Journal of Grid Computing (2004)
11. DataGrid Security Coordination Group: Security Requirements Testbed 1 Security Implementation. (2002) https://edms.cern.ch/document/340234.
12. IETF: PKIX Charter. (2004) http://www.ietf.org/html.charters/pkix-charter.html.
13. Butler, R., Engert, D., Foster, I., Kesselman, C., Tuecke, S., Volmer, J., Welch, V.: Design and deployment of a national-scale authentication infrastructure. IEEE Computer **33** (2000) 60–66
14. Tuecke, S., Welch, V., Engert, D., Pearlman, L., Thompson, M.: Internet X.509 Public Key Infrastructure Proxy Certificate Profile. (2003) http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-10.txt.
15. Housley, R., Polk, W., Ford, W., Solo, D.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. (2002) RFC 3280.
16. OpenSSL. (2004) http://www.openssl.org/.
17. Globus Simple CA. (2004) http://www.globus.org/security/simple-ca.html.
18. OpenCA. (2004) http://www.openca.org/.
19. Sun Open Network Environment. (2004) http://wwws.sun.com/software/sunone/.

20. Global Grid Forum. (2004) http://www.ggf.org/.
21. Butler, R., Genovese, T.: Global Grid Forum Certificate Policy Model. (2003)
22. Ball, E., Chadwick, D., Basden, A. In: The Implementation of a System for Evaluating Trust in a PKI Environment. Volume 2 of Evolaris. SpringerWein (2003) 263–279
23. NSF Middleware Initiative. (2004) http://www.nsf-middleware.org/.
24. Hanushevsky, A., Cowles, R.: Virtual Smart Card. (2002) http://www.slac.stanford.edu/ abh/vsc/.
25. European Grid Policy Management Authority for e-Science. (2004) http://www.eugridpma.org/.
26. Distributed European Infrastructure for Supercomputing Applications. (2004) http://www.deisa.org/.
27. South Eastern European Grid-enabled eInfrastructure Development. (2004) http://www.see-grid.org/.
28. GridPMA. (2004) http://www.gridpma.org/.

Grid Enabled Optimization

Hee-Khiang Ng, Yew-Soon Ong, Terence Hung[2], and Bu-Sung Lee[1]

[1] School of Computer Engineering, Nanyang Technological University,
Nanyang Avenue, Singapore 639798
{mhkng, asysong, ebslee}@ntu.edu.sg
[2] Institute of High Performance Computing, Singapore Science Park II,
Singapore 117528
terence@ihpc.a-star.edu.sg

Abstract. In this paper, we present a scalable parallel framework, which employs grid computing technologies, for solving computationally expensive and intractable design problems. Using an aerodynamic airfoil design optimization problem as an example the application of the grid computing strategies is discussed.

1 Introduction

Grid [1] Computing has gained widespread popularity in the research community for distributed and parallel computing because it has tremendous potential in enabling complex applications, especially those requiring huge amount of computation power and data storage requirements. A rising trend in the science and engineering world is in the utilization of increasingly high-fidelity and accurate analysis codes in the design analysis and optimization process. In many application areas such as photonics, electro-magnetics, aerospace, biomedical, micro-electro-mechanical systems and coupled-field multidisciplinary system design processes, the design process generally requires a Computational Structural Mechanics (CSM), a Computational Fluid Dynamics (CFD) or a Computational Electronics & Electro-magnetics (CEE) simulation procedure. The time taken for these processes generally varies from many minutes to hours or days of supercomputing time. This would often lead to high computing costs in the design optimization process, hence a much longer design cycle time to locate the optimum design solution. The benefits of Grid computing in the context of evolutionary computation, especially on computational expensive design optimization problem are numerous. In particular, the ability to tap on vast compute power. For instance, specialized high-fidelity analysis codes and computing nodes owned by different design teams that spans across geographically distributed locations may be shared and better utilized.

Solving large scale optimization problems requires a huge amount of computational power. The size of optimization problems that can be solved on a few CPUs has been limited due to a lack of computational power. The recent development in Grid has received much attention as a powerful and inexpensive way of solving large scale optimization problems that an existing single-unit CPU cannot process. The aim of

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 296 – 304, 2005.

this paper is to show that grid computing provides tremendous power to solve such large scale optimization problems.

The rest of this paper is organized as follows: In section 2, we present a brief overview of the fundamental concept of Evolutionary Algorithms, and in Section 3 we discuss the aerodynamic design and in section 4, we describe the implementation aspects of wrapping airfoil analysis code as services and realizing genetic algorithms as Grid enabled services. Finally in section 5, we conclude this paper.

2 Evolutionary Algorithms

In this section, we offer a brief overview on evolutionary algorithms, in particular, the general forms of parallel evolutionary algorithms that exist. Evolutionary Algorithms (EAs) are modern stochastic search techniques inspired by the 'survival of the fitness' principle of the Darwinian theory of natural evolution. By simulating natural evolution, EAs has been employed for solving many complex problems. A well-known strength of EAs is the ease of extensions to incorporate parallelism []. For instance, Parallel Genetic Algorithms (PGAs) are extensions of the standard Genetic Algorithm. Since the algorithm works with sets of populations, instead of a single individual, the basic concept of PGA is a simple division of the tasks in a classical GA across different processors. The other advantage of PGA is that it facilitates speciation, a process by which different populations evolve in different directions simultaneously. They have been shown to speed up the search process as well as to obtain higher quality solutions when dealing with complex design problems. In the paper, the PGA is applied to the optimization of an aerodynamic airfoil design problem, whereby the optimization is carried out through a number of generations. In each generation, the algorithm produces populations for the design analysis process. These results would then act as inputs in the next iterative stage of the numerical procedure.

Aerodynamic design optimization is one of the most frequently tackled problems in the aeronautics industry. It generally poses serious economic questions as the analysis is often very computationally expensive and often requires very intensive design solutions. In such situations, due to the large aerodynamic design search space often required, stochastic optimization algorithms such as EA are often employed, in the search for the near optimum design. Typically, such algorithms requires thousands of function evaluations, which often requires excessive CPU times, in locating the near optimal solution, whilst each function evaluation, using the high-fidelity analysis codes, may take up to many minutes to hours of computing time. Hence, it is not uncommon to hear that the design of an aircraft wing, using such algorithm, usually takes up to several months of CPU time even while running on supercomputers. This is a major obstacle in the practical application of EA optimization to complex engineering design problems, as the design process may prove to be computationally intractable if the optimal design is desired.

On the other extreme, however, the main advantage of EA is that it is a population-based technique, which allows the parallelization of the algorithm, in such a way that the designs in each generation can be evaluated simultaneously across the range of available machines. This is undertaken here as the computing complications encountered in the aerodynamic airfoil design problem are tackled with a parallelized

EA, deployed on a High Performance Computing (HPC) platform such as NetSolve, across multiple HPC clusters. This allowed a large number of these evaluation tasks to be farmed out to various nodes/CPUs within the cluster using a cluster scheduler arrangement. As a result, improved optimization solutions in the design cycle are achieved. The details of the numerical implementation of these algorithms are described in subsequent sections.

2.1 Genetic Algorithms

Genetic Algorithms is a member of the EA family employed in the numerical procedure. The term Genetic Algorithms or GA was first introduced by Holland in his seminal book referring to the probabilistic search techniques inspired by the 'survival of the fittest' principle of the Darwin theory of natural selection. Artificial creatures are created, which will then compete in a struggle for life. The fittest will then survive and are then allowed to reproduce. New creatures will then be created again using some operators such as crossover and mutation. The schematic workflow of this repetitive process is shown in Figure 1.

```
Procedure GA
Begin
  for i:=1 to n do begin
      initialise(x_i)          // with random values
      f(x_i):=evaluate(x_i)    // determine fitness
  end
  while termination condition not satisfied do begin
      p_1...n:= select from x_1...n in random order
      for i:=1 to n step 2 do begin
              x'_i,x'_i+1:=crossover(pi,pi+1)
              mutate(x'_i)
              mutate(x'_i+1)
      end
      x_1...n:=x'_1...n        // copy new population
      for i:=1 to n do begin
              f(x_i):=evaluate(x_i)
      end
    end {while}
end
```

Fig. 1. The Schematic Workflow of Canonical GA

3 Aerodynamic Airfoil Design

One of the most important features of an aircraft is during its takeoff, *i.e.* to get the body off the ground and subsequently to maintain it there. This usually comes from the efficient process of lifting the body up. By far the most common means of lifting, widely used in today's industry, is the wing. It is a concept ergonomically designed and embodied in current modern machinery. In actual fact, wings of all shapes and

sizes can be used to lift these bodies up, but special care must be taken to ensure that these wings are of the proper geometry. This is to ensure stability, efficiency and performance excellence.

Simplifications were made to the original design of the aerodynamic airfoil design problem. As the main emphasis of the study is to show the possibility of using EA, with the proposed alternative procedure, to solve such problems, the assumptions and changes made does not affect the gist of the problem. As such, the details of the problem are not discussed here, but can be found in literatures published elsewhere [8].

The simplified problem, considered in this study, is to achieve the best 24 design variables with an optimized drag profile, *i.e.* a minimized drag and a maximized lift, at a Mach number value of $M\propto \in [0.45,0.5]$, with the best midpoint value of 0.5 at an angle of attack (AOA) of 0.2. These requirements are schematically illustrated in Figure 2.

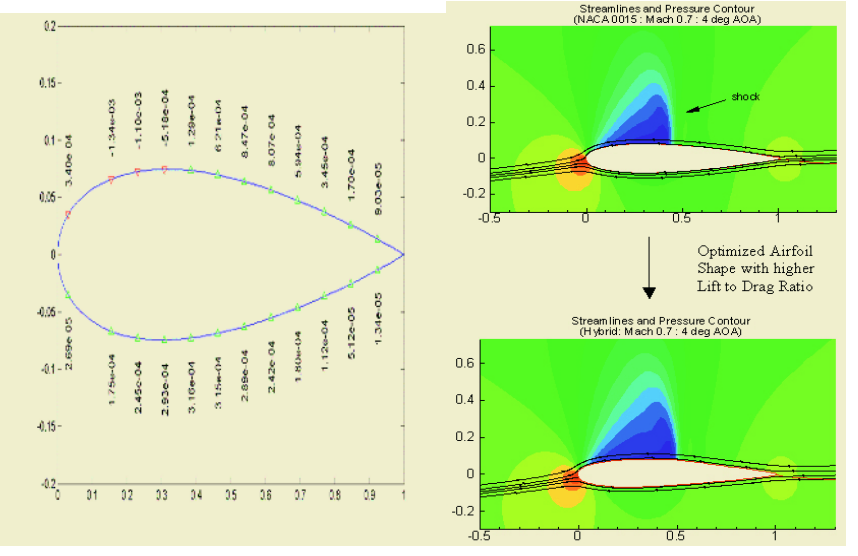


Fig. 2. Simplified aerodynamic airfoil design problem

4 Grid Enabled Optimization (GEO)

Here, the architecture of the grid enabled optimization services for each grid cluster, on the aerodynamic airfoil design optimization problem, is described. For this particular computing setup, the airfoil analysis problem enabled as an 'airfoil analysis' service is implemented to facilitate parallelism in the algorithm. This service were deployed on the cluster and registered to the resource Agent. This implies that the latter would search for the available resources and allocate them to the 'airfoil analysis' Services.

In the implementation of the proposed grid computing technology in this paper, two essential features are highlighted. The first is that a extended GridRPC API is used to 'gridify' the applications. This is done or 'gridified' using Globus [13] API, CoG built into our extension of the GridRPC API. This choice is down to its relative simplicity in implementation and its ability to offer high-level abstraction, *i.e.* the concealing of complex grid environment from the users.

The second feature of the proposed procedure is the implementation of a meta-scheduler, to be discussed later. From reviews conducted, it was discovered that the current state-of-the-art GridRPC implementations lack the feature for automatic resource discovery and selection on the grid environment. As a result, users have to perform look-up and select resources that suit their needs manually before they can assign computing tasks to the respective resources. This leads to the inefficient use of resources since large amount of resources exist on Grid and these are often dynamic. Furthermore, most optimization problems generally have a large number of evaluation tasks that requires many identical computations, with different analysis parameter sets. Hence it is extremely inefficient if the interactions between the Clients and resources are repeated many times for the same remote procedure call. This is overcome here with the use of a meta-scheduler which schedules GridRPC requests and balances the workload across multiple clusters on grid.

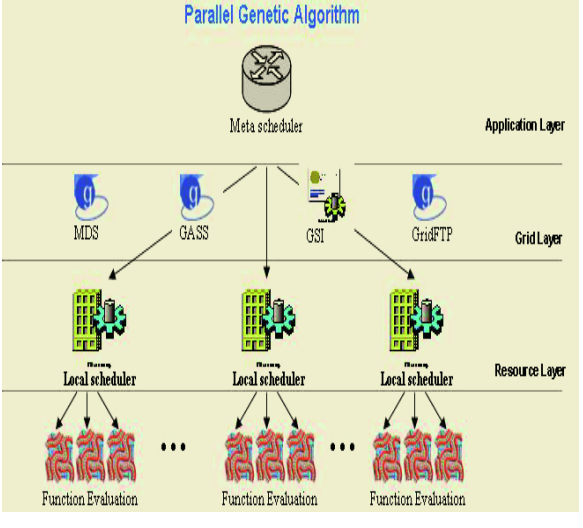


Fig. 3. Components in the GEO

Another characteristic of the proposed grid computing system is the employment of a Lightweight Directory Access Protocol (LDAP), to query the MDS database for available resources and workload information. To the GridRPC Client, this acts as a centralized directory service to scout for the available resources and identify their location within the grid environment. In its initialization process, the GridRPC function requires a simple entry from the MDS server and a virtual organization name to

retrieve the available resources and check whether or not these resources are capable of servicing further GridRPC requests. These checks were performed for software, *i.e.* remote procedure, specialized libraries, *etc*, hardware, *i.e.* platform, operating system, *etc* and the workload. In this regard, the Ganglia monitor toolkit is employed, to monitor and provide information relating to the available clusters in the computing systems.

For the multi-clusters execution of the PGA task, the adopted procedure is developed from the Globus toolkits. It basically involves using the Global Access Secondary Storage (GASS) to marshal the results back to the Client program, where the multi-clusters farming process initiates. GridFTP is then used to perform the transfer of the design input variables files to the selected clusters, which are available on grid. This is then followed by the use of Globus job submission protocol to facilitate the execution of the PGA process at the grid resources. This outline is shown in Figure 3, clearly presenting the different components in the procedure.

4.1 GEO Workflow

For the aerodynamic airfoil design optimization problem considered, the process begins with the meta-scheduler searching for the grid resources, to obtain the list of compute resources. The meta-scheduler also contains the information relating to the aerodynamic problem, which is deployed on the resources through the lookup on Globus MDS. Once the required computational resources are identified, the input design variables files are then transferred to the selected resources using GridFTP. Upon the successful transfer of these files to the resources, the application will then farm out the different chromosomes to the different clusters in parallel using the Globus execution command. Globus will then initiate and execute this task on the remote grid resources. When execution completes, the results are then marshalled back to the Globus Client application based on the GASS mechanism. The Client will then collect this data and continue with the GA program execution, *i.e.* iterating the program until the termination criterion is met, which in this report, is number of generations to run the analyses.

As discussed, the meta-scheduler is used to achieve immense parallelism in the search process as well as the seamless access to the grid resources. In the context of evolutionary algorithm, the term parallelism here meant the division of a population into isolated subpopulations. This gives rise to the concept of distributed EA, where each compute node may be allocated to evolve a subpopulation of individuals as well as the periodic migration of chromosomes among each another. Its implementation here is based on GridRPC [10], a remote procedure call standard interface for grid applications, proposed by the Grid Forum Advanced Programming Models research group of the Global Grid Forum [11].

Basically, the workflow of the Grid-enabled GA procedure in Figure 4 can be assembled into 9 main stages. These stages are as follows: -

1) The procedure begins with the Client contacting the meta-scheduler for services and resources necessary for the performance of the analyses.
2) The metascheduler, which in essence is the heart of the solution process, then obtains a list of the available resources together with their availability status. This is obtained from services such as the Globus Monitoring and Discovery

Service (Globus MDS) [12] and the Ganglia Monitoring Service (Ganglia) [13]. It then farms out the parallel analysis requests to the available grid resources based on the workload and the resource information retrieved from the monitoring services. The important point to take note here is that every time a new resource joins the grid, it is registered to the Globus MDS. This implies that as more computer resources and power are available, it can be added to the system.

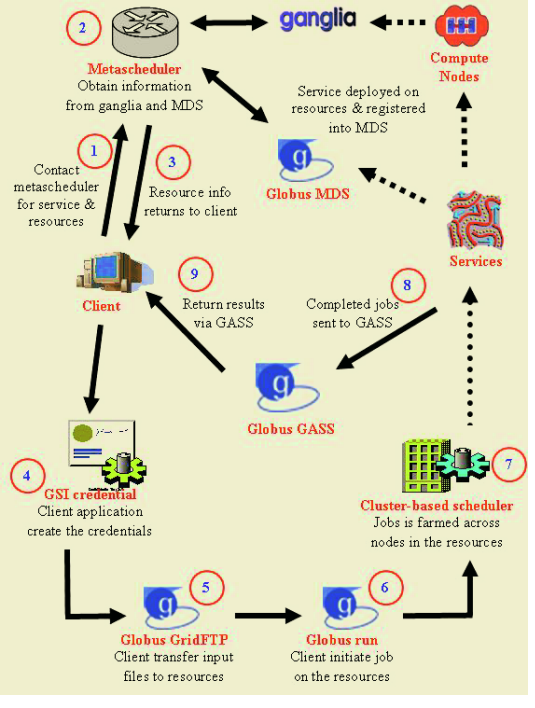


Fig. 4. GEO Workflow

3) These resources information and services are then fed back to the Client.
4) Upon obtaining the information in relation to the resources and services, the GSI [14] credentials are then generated. This can be viewed upon as an authentication or the authority to use the computing resources available in the system.
5) GridFTP [15] is then performed, to transfer the necessary input files from the Client to the resources and services.
6) The analysis job is then initiated at the resource cluster through the Globus submission protocol.
7) Whenever the Grid Resource Allocation Manager (GRAM) [16] gatekeeper of a cluster receives a service request, an instance of the service will be ini-

tialized on the master node of the cluster. Subsequently, the set of service requests will then be farmed across multiple computing nodes in the cluster by the cluster level scheduler, such as NetSolve. The responsibility of the local agent in each cluster is to perform local scheduling and resource discovery across computing nodes in the cluster for basic services.

8) The results of the completed jobs are then staged on the GASS [17].
9) Upon which, the results are channeled back from the GASS to the Client for assessment and evaluation.

5 Conclusions

The aim of this paper is to present our grid computing platform, which employs grid computing technologies, in solving computationally expensive and intractable design optimization problems. We successfully deployed the airfoil design optimization problem, onto the Nanyang campus grid [18]. From which, the optimization problem is executed across the grid computing platform, unleashing and utilizing the full power of grid computing technology.

References

1. SIGHT, http://www.engineous.com, Engineous software Inc, "*iSIGHT, Version 7.1*", 2004
2. I.Foster, C.Kesselman, and S.Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations" , *International J.Supercomputer Applications*, vol. 15, no. 3, 2001
3. Agrawal S., Dongarra J., Seymour K., Vadhiyar S., "NetSolve: past, present, and future; a look at a grid enabled server", 2002.
4. Arnold D. C., Casanova H., Dongarra J., "Innovations of the NetSolve grid computing system", 2002.
5. The Globus Alliance, http://www.globus.org/, 2004
6. Goldberg D. E., "*Genetic algortihms in search, optimisation and machine learning*", 1989.
7. Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane, Nell Rehn, Mike Russell. Designing Grid-based Problem Solving Envionments and Portals Argonne National Laboratory, Argonne, IL 60439.
8. Giannakoglou K. C., "Design of optimal aerodynamic shapes using optimization methods and computational intelligence", Process in Aerospace Sciences, Vol 38, pp 43-76, 2002.
9. Ho Q. T., Ong Y. S., Cai W. T., "Gridifying aerodynamic design problem using GridRPC", 2nd International Workshop on Grid and Cooperative Computing, Shanghai, China, 2003.
10. Nakada H., Matsuoka S., Seymour K., Dongarra J., Lee C., Casanova H., "GridRPC: A remote procedure call API for grid computing", Grid Computing – Grid 2002, LNCS 2536, pp 274-278, 2002.
11. http://www.ggf.org/, GGF, 2004.
12. Fitzgerald S., Foster I., Kesselman G., Laszewski V., Smith W., Tuecke S., "A directory service for configuring high-performance distributed computations", Proceedings 6th IEEE Symposium on High-Performance Distributed Computing, pp 365-375, 1997.
13. http://sourge.ganglia.net, 2004
14. http://www-unix.globus.org/security/, The Globus GSI, 2004.
15. http://www.globus.org/datagrid/gridftp.html, GridFTP, 2004.

16. The Globus Resource Allocation Manager (GRAM), http://www-unix.globus.org/developer/resource-management.html, 2004,
17. http://www.globus.org/gass/, Global access to secondary storage (GASS), 2004.
18. Nanyang Campus Grid, http://www.ntu-cg.ntu.edu.sg, 2004

Towards a Coordination Model for Parallel Cooperative P2P Multi-objective Optimization*

M. Mezmaz, N. Melab, and E.-G. Talbi

LIFL, CNRS UMR 8022,
INRIA Futurs - Dolphin,
Université des Sciences et Technologies de Lille,
59655 - Villeneuve d'Ascq cedex - France
{mezmaz, melab, talbi}@lifl.fr

Abstract. Existing Dispatcher-Worker Peer-to-Peer (P2P) computing environments are well-suited for multi-parameter applications. However, they are limited regarding the parallel computing where the generated tasks need to communicate. In this paper, we investigate that limitation and propose a coordination model for parallel P2P multi-objective optimization (MOO). The model has been implemented on top of the XtremWeb middleware. Then, it has been experimented on a combinatorial optimization application: a parallel branch-and-bound algorithm applied to the multi-objective (MO) Flow-Shop scheduling problem. The preliminary results obtained on a network of 120 heterogeneous PCs demonstrate the efficiency of the proposed approach.

Keywords: P2P Computing, Parallelism and Coordination, Multi-objective Optimization, Branch-and-Bound, Flow-Shop.

1 Introduction

In many domains such as telecommunications, genomics, transport, and so on, the tackled optimization problems need more and more computational power. However, very often the users do not have high-end supercomputers to deal with these problems. Therefore, they have to scale down the size of the problems to solve them. In the last decade, Peer-to-Peer (P2P) computing [1] has become a real alternative to traditional supercomputing environments for the development of parallel applications that harness massive computational resources.

Nowadays, existing Dispatcher-Worker middlewares such as SETI@HOme [2], XtremWeb [3] and JNGI/JXTA [4] facilitate the development of parallel applications on P2P systems. They include a Dispatcher (server) that maintains a list of work unites and their associated data, and a set of workers (volunteer peers) that steal these work units according to the cycle stealing model. These environments all well-suited for multi-parameter applications that can be naturally

* This work is a part of the current national joint grid project GGM of ACI-Masse de Données.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 305–314, 2005.

split into several independent tasks. However, they are not adapted to parallel distributed applications where communications between peers are needed.

In this paper, we investigate this issue by proposing a data-driven coordination model which provides communication through a tuple space. A tuple space is a global associative memory consisting of a bag (or multi-set) of tuples. The model is dedicated to support parallel multi-objective optimization (MOO) on top of Dispatcher-Worker systems such as XtremWeb. The proposed model extends Linda [5] with group and non-blocking coordination operations that are very useful for P2P multi-objective optimization. The model has been implemented as a software layer on top of XtremWeb. At implementation level, the coordination is based on Java RMI calls.

The model has been experimented on a combinatorial optimization application: a parallel branch-and-bound algorithm applied to the bi-objective Flow-Shop scheduling problem. The problem consists in scheduling in the same order a set of jobs on a set of machines, such that the total lateness (tardiness objective) and the total completion time (makespan) are minimized. The parallelism consists in exploring in parallel a large irregular tree. The work units distributed to the workers are sub-trees to be explored. The experimentations have been conducted on a network of 120 heterogeneous PCs during more than two full days. The preliminary results demonstrate the efficiency of the proposed model and its implementation.

The rest of the paper is organized as follows: Section 2 presents a brief overview on P2P computing and coordination. Section 3 highlights the requirements of parallel cooperative MOO and then describes the proposed coordination model. Thereafter, its implementation on top of the middleware XtremWeb is discussed. Section 4 presents the experimentation of the model through the parallel Branch-and-Bound applied to the Bi-objective Flow-Shop Scheduling problem, and analyzes the preliminary experimental results. Finally, Section 5 draws some concluding remarks and the perspectives of the presented work.

2 P2P Computing and Coordination

Nowadays, there exist several fully distributed P2P systems meaning they do not include a central server [6]. These systems are often well-suited for the storage of massive data sets and their retrieval. There are also P2P systems dedicated to large scale computing [3, 2, 4, 7], but only few of them are fully distributed [7]. Fully distributed computing P2P are just emerging and are not yet mature nor stable to be exploited.

More mature software systems such as XtremWeb[3] and SETI@Home[2] are today those based on a Dispatcher-Worker architecture. In such systems, clients can submit their jobs to the Dispatcher. A set of volatile workers (peers) request the jobs from the Dispatcher according to the cycle stealing model. Then, they execute the jobs and return the results to the Dispatcher to be collected by the clients. In these middlewares, even a central server (the Dispatcher) is required

for controlling the peers (workers) they are considered as P2P software environments. Indeed, an important part of these systems is executed on these peers with a high autonomy. In addition, a hierarchical design allows them to deal with a larger number of peers.

One of the major limitations of P2P computing environments is that they are well-suited for embarrassingly parallel (e.g. multi-parameter) applications with independent tasks. In this case, no communication is required between the tasks, and thus peers. The deployment of parallel applications needing cross-peer/task communications is not straightforward. The programmer has the burden to manage and control the complex coordination between the workers. To deal with such issue existing middlewares must be extended with a software layer which implements a coordination model. Several interesting coordination models have been proposed in the literature [8]. In this paper, we focus only on one of the most popular of them i.e. Linda [5] because our proposed model is inspired from that model.

In the Linda model, the coordination is performed through generative communications. Processes share a virtual memory space called a *tuple-space* (set of tuples). The fundamental data unit, a tuple, is an ordered vector of typed values. Processes communicate by reading, writing, and consuming these tuples. A small set of four simple operations allows highly complex communication and synchronization schemes:

- *out(tuple)*: puts *tuple* into *tuple-space*.
- *in(pattern)*: removes a (often the first) tuple matching *pattern* from *tuple-space*.
- *rd(pattern)*: is the same as *in(pattern)*, but does not remove the tuple from *tuple-space*.
- *eval(expression)*: puts *expression* in *tuple-space* for evaluation. The evaluation result is a tuple left in *tuple-space*.

Due to the high communication delays in a P2P system, tuple rewriting is very important as it allows to reduce the number of communications and the synchronization cost. Indeed, in Linda a rewriting operation is performed as an "in" or "rd" operation followed by a local modification and an "out" operation. The operations "in"/"rd" and "out" involve two communications and an heavy synchronization. Therefore, a rewriting (or update) operation is very useful for coordination in P2P environments.

3 The Proposed P2P Coordination Model

3.1 Parallel MOO and Coordination

A multi-objective problem (MOP) consists generally in optimizing a vector of nb_{obj} objective functions $F(x) = (f_1(x), \ldots, f_{nb_{obj}}(x))$, where x is an d-dimensional decision vector $x = (x_1, \ldots, x_d)$ from some universe called *decision space*. The space the objective vector belongs to is called the *objective space*.

F can be defined as a cost function from the decision space to the objective space that evaluates the quality of each solution $(x_1, \ldots, x_d)$ by assigning it an objective vector $(y_1, \ldots, y_{nb_{obj}})$, called the *fitness*.

Unlike single-objective optimization problems, a MOP may have a set of solutions known as the *Pareto optimal set* rather than an unique optimal solution. The image of this set in the objective space is denoted as *Pareto Front or PF*). Graphically, a solution x is Pareto optimal if there is no other solution x' such that the point $F(x')$ is in the dominance cone of $F(x)$. This dominance cone is the box defined by $F(x)$, its projections on the axes and the origin (Fig. 1).

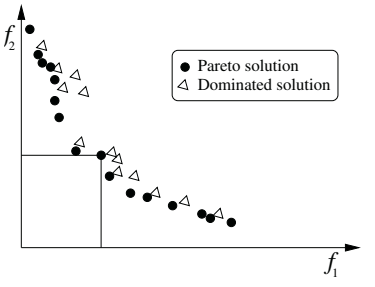


Fig. 1. Example of Pareto solutions

There are two major categories of MOO methods: MO exact methods and MO meta-heuristics. While the first category allows to find optimal solutions for MOPs, the second class provides near-optimal solutions in a reasonable time. Real-world MOPs involve highly constrained design and and high computational cost. Parallelism is proved to be a powerful way to achieve efficiency and effectiveness. In general, parallel MO exact methods consist in exploring in parallel a search three. The processes share the best found PF, which is remotely updated each time a process has discovered in its search sub-tree a better PF. Coordination operations are required to update this shared information.

In parallel models of MO meta-heuristics [9] such as the island model, different processes cooperate by exchanging Pareto optimal solutions in order to improve the effectiveness. The exchange may be performed either directly by message passing or through a shared space. In the last case, a coordination model is required to ensure the cooperation.

3.2 The Coordination Model

Designing a coordination model for parallel MOO requires the specification of the content of the tuple space, a set of coordination operations and a pattern matching mechanism. According to the comments of the previous sub-section, the tuple space may be composed of a set of Pareto optimal solutions and their

corresponding solutions in the objective space. For the parallel exact MO methods, all the solutions in the tuple space belong to the same PF i.e. the actual best found one. For the parallel island model of the MO meta-heuristics, the tuple space contains a collection of (parts of) Pareto optimal sets deposited by the islands for migration. The mathematical formulation of the tuple space (*Pareto Space or PS*) is the following:

$$PS = \bigcup PO,\ with\ PO = \{(x, F(x)),\ x\ is\ Pareto\ optimal\}$$

In addition to the operations provided in Linda, parallel P2P MOO needs other operations. These operations can be divided in two categories: *group* operations and *non-blocking* operations. Group operations are useful to manage multiple Pareto optimal solutions. *Non-blocking* operations are necessary to take into account the volatile nature of P2P systems. In our model, the coordination primitives are defined as follows:

- *in, rd, out and eval*: These operations are the same as those of Linda defined in Section 2.
- *ing(pattern)*: Withdraws from PS all the solutions matching the specified pattern.
- *rdg(pattern)*: Reads from PS a copy of *all* the solutions matching the specified pattern.
- *outg(setOfSolutions)*: Inserts multiple solutions in PS.
- *update(pattern, expression)*: Updates *all* the solutions matching the specified pattern by the solutions resulting from the evaluation of *expression*.
- *inIfExist, rdIfExist, ingIfExist and rdgIfExist*: These operations have the same syntax than respectively *in, rd, ing and rdg* but they are *non-blocking* probe operations.

The *update* operation allows to locally update the Pareto space, and so to reduce the communication and synchronization cost. The pattern matching mechanism depends strongly on how the model is implemented, and in particular on how the tuple space is stored and accessed. For instance, if the tuple space is stored in a database the mechanism can be the request mechanism used by the database management system. More details on the pattern matching mechanism of our model are given in the next Section.

3.3 Implementation on Top of XtremWeb

XtremWeb [3] is a Java P2P project developed at Paris-Sud University. It is intended to distribute applications over a set of peers, and is dedicated to multi-parameter applications that have to be computed several times with different inputs. XtremWeb manages tasks following the Dispatcher-Worker paradigm. Tasks are scheduled by the Dispatcher to workers only on their specific demand since they may adaptively appear (connect to the Dispatcher) and disappear (disconnect from the Dispatcher). The tasks are submitted by either a client or a worker, and in the latter case, the tasks are dynamically generated for parallel execution. The final or intermediate results returned by the workers are stored in a MySQL database. These results can be requested later by either the clients

or the workers. The database stores also different information related to the workers and the deployed application tasks.

XtremWeb is well-suited for embarrassingly parallel applications where no cross-peer communication occurs between workers, and these can only communicate with the Dispatcher. Yet, many parallel distributed applications particularly parallel MOO ones need cooperation between workers. In order to free the user from the burden of managing himself or herself such cooperation we propose an extension of the middleware with a software layer.

The software layer is an implementation of the proposed model composed of two parts: a coordination API and its implementation at the worker level and a coordination request broker (CRB). The Pareto Space is a part of the MySQL database associated with the Dispatcher. Each tuple or solution of the Pareto Space is stored as a record in the database. From the worker side the coordination API is implemented in Java and in C/C++. The C/C++ version allows the deployment and execution of C/C++ applications with XtremWeb (written in Java). The coordination library must be included in these programmer applications. From the Dispatcher side, the coordination API is implemented in Java as a Pareto Space manager. The *CRB* is a software broker allowing the workers to transport their coordination operations to the Dispatcher through RMI calls.

4 Application to Parallel MO Branch-and-Bound

In this Section, we describe the sequential B&B algorithm, then a parallel version using the proposed coordination model, and finally some experiments performed on a P2P network through the Flow Shop problem.

4.1 Parallel MO Branch-and-Bound

MO Branch-and-bound algorithms (MO-B&B) solve MOPs by iteratively partitioning the solution space into subspaces (each subspace is associated with a sub-MOP). In this paper, we assume that the MOP to solve is a minimization MOP. A sequential MO-B&B algorithm consists in iteratively applying five basic operations over a list of problems: *Branching*, *Resolution*, *Bounding*, *Selection* and *Elimination*.

Successive branching (decomposition) operations create a tree of MOPs rooted in the original MOP. The value of the best PF found so far is used to prune the tree and eliminate the MOPs that are likely to lead to worse solutions. At each step, a MOP is selected either according to the bound values (as in *best-first* strategy) or not (as in *depth-first* and *breadth-first* strategies). The selected MOP may not be split because it has no solution or because a solution is already be found. In this case, it is solved, and if its solution can improve the best known PF this latter is updated. If the MOP can be split than it is decomposed into smaller sub-MOPs. A sub-MOP is eliminated if its bound value is not better that the best known PF. Otherwise, it is added to the pool of MOPs to be solved.

There exist different parallel models in B&B algorithms [10]. We focus here on the most general approach, in which the B&B tree is built in parallel by performing simultaneously the operations presented above on different sub-MOPs. According to such approach, the design of parallel B&B algorithms is based on three major parameters: the *execution mode*, the *work sharing strategy* and the *information sharing policy*. The execution mode defines what processes do after completion of a work unit, and may be *synchronous* or *asynchronous*. The processes (do not) wait for each other in a(n) (a)synchronous mode. The work sharing strategy defines how work units are assigned to processes to efficiently exploit available parallelism. The information sharing policy indicates how the best-known solution is published and updated.

In this paper, we propose an asynchronous parallel cooperative Dispatcher-Worker MO B&B algorithm. Asynchrony is required by the heterogeneity nature of the P2P target execution architecture. The work sharing strategy follows the idle cycle or work stealing paradigm. Each worker maintains its local pool of MOPs to be solved. When the local pool is empty, the worker sends a work request to the Dispatcher. The information sharing issue is solved as the following: the best known PF is published and maintained by the Dispatcher. This information is requested by the workers, and updated each time a better PF is locally found.

At each step, the worker tests if there is some MOP to solve in its local work pool. If the pool is empty it requests work from the Dispatcher. The Dispatcher replies with a pool of work units and the value of the best-known PF. This value is stored locally. Otherwise, if there is some work in the local pool, the worker performs a step of the sequential MO-B&B on its local pool. Thereafter, it probably requests the Dispatcher to update the best-known PF by merging this latter with its local version. The operation is performed by the Pareto Space Manager on the Dispatcher. The new best-known PF is returned to the calling worker.

4.2 Application to the Flow-Shop MOP

The Flow-Shop MOP is one of the numerous scheduling MOPs [11] that has received a great attention given its importance in many industrial areas. The MOP can be formulated as a set of N jobs $J_1, J_2, \ldots, J_N$ to be scheduled on M machines. The machines are critical resources as each machine can not be simultaneously assigned to two jobs. Each job J_i is composed of M consecutive tasks $t_{i1}, \ldots, t_{iM}$, where t_{ij} represents the j^{th} task of the job J_i requiring the machine m_j. To each task t_{ij} is associated a processing time p_{ij}, and each job J_i must be achieved before a due date d_i.

The MOP being tackled here is the Bi-objective Permutation Flow-Shop MOP (*BPFSP*) where jobs must be scheduled in the same order on all the machines. Therefore, two objectives have to be minimized: (1) C_{max}: Makespan (Total completion time), (2) T: Total tardiness. The task t_{ij} being scheduled at time s_{ij}, the two objectives can be formulated as follows:

$$f_1 = C_{max} = Max\{s_{iM} + p_{iM} | i \in [1 \dots N]\}$$
$$f_2 = T = \sum_{i=1}^{N} [max(0, s_{iM} + p_{iM} - d_i)]$$

In this paper, we do not focus on how the MO-B&B technique is applied to BPFSP, the reader is referred to [12] for such details. We are interested in the parallel P2P design features. In the implementation of the parallel MO-B&B applied to BPFSP, the best-known PF is updated if a sufficient number of iterations is already performed. The adopted selection strategy is the *depth-first* one, the node with the best bound being chosen at each step. The *update* coordination operation is executed on the Dispatcher by the Pareto Space Manager. First, it consists in performing an union between the two sets: the global best-known PF (stored in the Pareto Space) and its local version. The new best-known PF is then selected from this union set by considering all the non-dominated solutions. The new result is returned to the calling Worker.

After a fixed number of iterations, if a Worker has work in its local pool it splits it into as many pools as available workers (considering itself). The Worker saves a pool for its own need, and submits (in a client role) the other pools to the Dispatcher through the *eval* coordination operation. The Dispatcher puts these work units in its task pool to be sent to available workers at their request.

4.3 Experimentation

The application has been deployed on the education network of the *Polytech'Lille* engineering school. The experimentation hardware platform is composed of 120 heterogeneous Linux Debian PCs. The BPFSP MOP benchmark is composed of 10 jobs and 20 machines ($n = 10$ and $m = 20$). The experimental results are summarized in Table 1. The total execution time is measured for the sequential and parallel versions. The execution time of the sequential version is normalized as the whole target architecture is heterogeneous. The machine where the sequential algorithm is executed is an Intel Pentium 4, 3 GHz, and is considered as a reference machine for the computation of the normalized factor. The normalized factor for each peer is obtained by using an application specific benchmark task that is sent to all workers that join the computational pool. The speed at which the benchmark task is completed is considered as the normalized factor.

Formally, let t_{ref} and t_i be the execution time of the benchmark on respectively the reference machine and the machine number $i = 1..N$ of the pool of worker peers. The normalized factor α_i associated with the worker peer i is computed as follows: $\alpha_i = \frac{t_i}{t_{ref}}$. Let α_{av} be the average normalized factor for all the worker peers. It can be formulated as: $\alpha_{av} = \frac{\sum_{i=1}^{N} \alpha_i}{N}$. The sequential time reported in Table 1 is the time obtained on an average peer, obtained by multiplying the sequential time obtained on the reference peer by the average factor.

The results show that the execution time is divided by over 29 on 120 machines. One has to note that the experiments have been performed during a working day, thus the experimentation environment is non-dedicated. On the

Table 1. Parallel MO-B&B vs. Sequential MO-B&B

	Sequential B&B	**Parallel B&B**
Total number of tasks	1	657
Total execution time	54h51	1h53

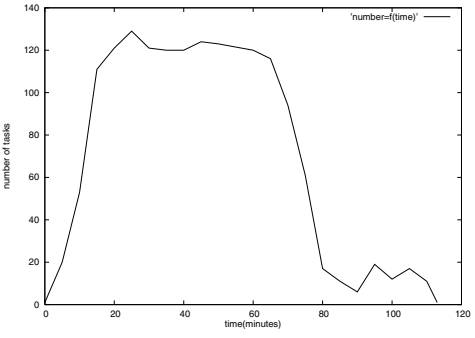


Fig. 2. Task generation over time

other hand, as Fig. 2 illustrates it, due to the nature of the application sufficient parallelism (for the 120 peers) is generated during only 1 hour over about two hours of total execution.

5 Conclusion and Future Work

In this paper, we have presented a coordination model for parallel cooperative MOO applications in a P2P environment. The model allows to overcome a major limitation of existing Dispatcher-Worker middlewares: they do not allow a straightforward communication between tasks executed by different peers. The model has been implemented on top of XtremWeb, a middleware dedicated to the execution of independent multi-parameter applications. The result is that the users can develop parallel cooperative applications in a transparent way. They do not need to manually manage and control the cooperation. Furthermore, the model is generic and can be integrated into another P2P computing middleware and applied in the context of another application domain.

The model has been experimented and validated on an MOO application: a parallel B&B algorithm applied to the Bi-criterion Permutation Flow-Shop Scheduling problem. The experimental results show that the time wasted by the PCs of the experimentation hardware platform is well exploited as the total execution time of the application is divided by a factor of 4 in a non-dedicated execution environment.

In the future, we will experiment and extend the model to deal with parallel cooperative meta-heuristics. We will also deploy it on top of another middleware (JNGI/JXTA [4]) to demonstrate its generic nature.

References

1. Oram, A.: Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates (2001)
2. Anderson, D., Cobb, J., Korpela, E., , Lepofsky, M., Werthimer, D.: SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM **Vol. 45** (2002) 56–61
3. Fedak, G., Germain, C., Neri, V., Cappello, F.: XtremWeb: building an experimental platform for Global Computing. Workshop on Global Computing on Personal Devices (CCGRID2001), IEEE Press (2001)
4. Verbeke, J., Nadgir, N., Ruetsch, G., Sharapov, I.: Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. In Proc. of the Third Intl. Workshop on Grid Computing (GRID'2002), Baltimore, MD (2002) 1–12
5. Gelernter, D.: Generative Communication in Linda. ACM Transactions on Programming Languages and Systems **Vol. 7** (1985) 80–112
6. Ripeanu, M.: Peer-to-Peer Architecture Case Study: Gnutella Network. 1^{st} IEEE Intl. Conf. on Peer-to-peer Computing (P2P2001) (2001)
7. Oliveira, L., Lopes, L., Silva, F.: P^3: Parallel Peer to Peer - An Internet Parallel Programming Environment. Intl. Workshop on Peer-to-Peer Computing, Pisa, Italy (2002)
8. Papadopoulos, G., Arbab, F.: Coordination models and languages. Advances in Computers: The Engineering of Large Systems, Academic Press **46** (1998)
9. van Veldhuizen, D., Zydallis, J., Lamont, G.: Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Trans. on Evolutionary Computation **7** (2003) 144–173
10. Gendron, B., Crainic, T.: Parallel branch-and-bound algorithms: Survey and synthesis. Operations Research **42** (1994) 1042–1066
11. T'kindt, V., Billaut, J.C.: Multicriteria Scheduling - Theory, Models and Algorithms. Springer-Verlag (2002)
12. Basseur, M., Lemesre, J., Dhaenens, C., Talbi, E.G.: Cooperation between Branch and Bound and Evolutionary Approaches to solve a BiObjective Flow Shop Problem. Worshop on Evolutionary Algorithms (WEA'04) (2004) 72–86

A Grid-Oriented Genetic Algorithm*

J. Herrera[1], E. Huedo[2], R.S. Montero[1], and I.M. Llorente[1,2]

[1] Departamento de Arquitectura de Computadores y Automática,
Universidad Complutense, 28040 Madrid, Spain
[2] Laboratorio de Computación Avanzada,
Simulación y Aplicaciones Telemáticas,
Centro de Astrobiología (CSIC-INTA),
28850 Torrejón de Ardoz, Spain

Abstract. Genetic algorithms (GAs) are stochastic search methods that have been successfully applied in many search, optimization, and machine learning problems. Their parallel counterpart (PGA, parallel genetic algorithms) offers many advantages over the traditional GAs, such as speed, ability to search on a larger search space, and less likely to run into a local optimum. With the advent of Grid computing, the computational power that can be deliver to the applications have substantially increased, and so PGAs can potentially benefit from this new Grid technologies. However, because of the dynamic and heterogeneous nature of Grid environments, the implementation and execution of PGAs in a Grid involve challenging issues. This paper discusses the distribution of a PGA across the Grid using the DRMAA standard API and the Grid *W*ay framework. The efficiency and reliability of this schema to solve the One Max problem is analyzed in a globus-based research testbed.

1 Introduction

Genetics algorithms are search algorithms inspired in natural selection and genetic mechanisms. GAs use historic information to find new search points and reach an optimal problem solution. In order to increase the speed and the efficiency of sequential GAs, several Parallel Genetic Algorithm (PGA) alternatives have been developed. PGAs have been successfully applied in previous works, (see for example [1]), and in most cases, they succeed to reduce the time required to find acceptable solutions.

Traditionally, PGAs have tried to efficiently exploit the performance offered by massively parallel processing systems (MPPs). The development of the Grid has opened up avenues that could lead to a dramatic increase in performance of PGAs, in terms of execution time and problem size. However, the execution and development of Grid applications requires a high level of expertise and a significant amount of effort.

* This research was supported by Ministerio de Ciencia y Tecnología through the research grant TIC 2003-01321 and Instituto Nacional de Técnica Aeroespacial.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 315–322, 2005.

The main difficulties arise from the characteristics of the Grid itself, namely: complexity, heterogeneity, dynamism and high fault rate. To overcome these difficulties, we have developed a Globus submission framework, Grid*W*ay [2], that allows an easier and more efficient execution of jobs on a dynamic Grid environment. Grid*W*ay automatically performs all the job scheduling steps [3] (resource discovery and selection, and job preparation, submission, monitoring, migration and termination), provides fault recovery mechanisms, and adapts job execution to the changing Grid conditions [4].

On the other hand, the Grid lacks of a standard programming paradigm to port existing applications among different environments. The DRMAA (Distributed Resource Management Application API) [5] specification, developed within the *Global Grid Forum* (GGF)[1] framework, tries to fill this gap. The DRMAA specification constitutes a homogeneous interface to different DRMS (Distributed Resource Management Systems) to handle job submission, monitoring and control, and retrieval of finished job status.

In this work we analyze the distribution of a PGA across the Grid using the DRMAA standard API and the Grid*W*ay framework. In particular, the PGA considered here adopts a modified version of the fully connected multi-deme approach, to adapt it to the Grid characteristics mentioned above. The rest of the paper is organized as follows, Section 3 briefly reviews the main strategies proposed in the literature to distribute GAs. Then, we propose in Section 3.1 a Grid-aware distribution scheme of GAs, and we describe its implementation with DRMAA. The performance of this scheme in a research testbed is then discussed in Section 4. The paper ends with some conclusions.

2 Parallel Genetic Algorithms

The different alternatives proposed in the literature to parallelize genetic algorithms can be classified in three main categories [6]:

- **Single Population (Panmitic GA).** This kind of GAs is usually implemented using a Master/Worker paradigm [7]. Panmitic GAs can be efficiently used when the evaluation function requires a considerable amount of computational work. The main advantage of this method is that the search behavior of the sequential GA is not altered, and therefore all the available GA theory can be applied directly. However this approach is not well suited for a Grid because of the high network requirements of its communication pattern.
- **Single Population (Fine Grain) GA.** This type of GA has only one population and its spatial structure limits the interactions between individuals. This limit can be imposed at the chromosome level (each member can only interact with their neighbors) or at the population level (only member of the same subpopulation may mate during crossover).

[1] http://www.gridforum.org (2004)

– **Coarse Grain GA.** The main population is divided into subpopulations (demes), each one is independently evaluated in a different node. Probably the most important characteristic of these algorithms is the communication topology used to exchange information between subpopulations. The possible communication patterns include ring model (processes can only interact with their neighbors in a ring topology), master-slave (slave processes swap best individuals with the master) or all-to-all (all process swap best individuals with the others).
This kind of GA is difficult to understand because of the effects of migrations between populations are not fully understood. Moreover, coarse grain GAs introduce fundamental changes in the implementation of a simple GA.

In this work we will use a modified version of the coarse grain approach (see Section 3), since this algorithm does not imply a tightly coupled deme topology. Therefore it is more tolerant to the high latencies and dynamic bandwidths that can be expected in the Internet, unlike the single population alternatives.

3 Grid-Oriented Coarse Grain Genetic Algorithms

In order to develop efficient Grid-oriented genetic algorithms (GOGAs, following the notation introduced by H. Imade et al. [8]), the dynamism and heterogeneity of a Grid environment must be considered. In this way, traditional load-balancing techniques could lead to a performance slow-down, since, in general the performance of each computing element can not be guaranteed during the execution. Moreover, some failure recovery mechanisms should be included in such a faulty environment.

3.1 Algorithm Description

Taking into account the above considerations we will use a fully connected multi-deme genetic algorithm. In spite of this approach represents the most intense communication pattern (all demes exchange individuals every generation), it does not imply any overhead since the population of each deme is used as checkpoint files, and therefore transferred to the client in each iteration.

The initial population is uniformity distributed among the available number of nodes, and then a sequential GA is locally executed over each subpopulation. The resultant subpopulations are transferred back to the client, and worst individuals of each subpopulation are exchanged with the best ones of the rest. Finally, a new population is generated to perform the next iteration [6]. The schema of this algorithm is depicted in figure 1.

However, the previous algorithm may incur in performance losses when the relative computing power of the nodes involved in the solution process greatly differs, since the iteration time is determined by the slowest machine. In order to prevent these situations we allow an *asynchronous* communication pattern between demes. In this way, information exchange only occurs between a fixed

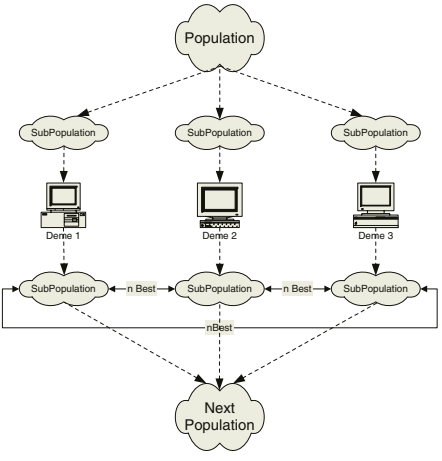


Fig. 1. Schema of fully-connected multi-deme genetic algorithm, with three computing nodes

number of demes, instead of synchronizing the execution of all subpopulations. The minimum number of demes that should communicate in each iteration depends strongly on the numerical characteristics of the problem. We will refer to this characteristic as *dynamic connectivity*, since the demes that exchange individuals differs each iteration.

3.2 Distributed Resource Management Application API

The Distributed Resource Management Application API (DRMAA) is an API specification for job submission, monitoring and control that provides a high level interface with Distributed Resource Management Systems (DRMS). In this way, DRMAA could aid scientists and engineers to express their computational problems by providing a portable direct interface to DRMS.

Although the DRMAA standard can help in exploiting the intrinsic parallelism found in some application domains, like GAs, the underlying DRMS is responsible for the efficient and robust execution of each job. The following aspects are considered by the Grid*W*ay framework, used in this work:

- Given the dynamic characteristics of the Grid, the Grid*W*ay framework periodically adapts the schedule to the available resources and their characteristics [4].
- The Grid*W*ay framework also provides adaptive job execution to migrate running applications to more suitable resources. So improving application performance by adapting it to the dynamic availability, capacity, cost of Grid resources, or its own requirements and preferences [4].
- Grid*W*ay also provides the application with fault tolerance capabilities by capturing GRAM callbacks, by periodically probing the GRAM *jobmanager*, and by inspecting the output of each job.

Table 1. Implementation of the Grid-oriented coarse-grain genetic algorithm using the DRMAA standard

```
//Initialize a new DRMAA session.
rc = drmaa_init (contact, error)
//Execute all jobs consecutively
for (i=0; i < ALL_JOBS; i++)
   rc = drmaa_run_job(job_id, jt, err_diag)
//Execute GOGA if it dosen't rise objective_function
while (!this->objective_function()){
 //Wait for (dynamic connectivity degree) jobs
 //and store results
 for (i=0; i < NUM_JOBS; i++)
  rc = drmaa_wait(job_id, &stat, timeout, rusage, err_diag)
 this->store_results())
 //Execute (dynamic connectivity degree) jobs consecutively
 for (i=0; i < NUM_JOBS; i++)
  rc = drmaa_run_job(job_id, jt, err_diag)
}
//Finalize DRMAA session.
rc = drmaa_exit(err_diag)
```

In particular, the following list describes the DRMAA interface routines implemented within the Grid*W*ay framework (see [9] for a detailed description of the C API):

- Initialization and finalization routines: Initialize and finalize a DRMAA session.
  - `drmaa_init`. Initialize DRMAA API library and create a new DRMAA session.
  - `drmaa_exit`. Disengage from DRMAA library and allow the DRMAA library to perform any necessary internal cleanup.
- Job template routines: These routines enable the manipulation of job definition entities to set parameters such as the executable.
  - `drmaa_set_attribute`. Adds ('name', 'value') pair to list of attributes in job template.
  - `drmaa_allocate_job_template`. Allocate a new job template.
  - `drmaa_delete_job_template`. Deallocate a job template. This routine has no effect on jobs.
- Job submission routines:
  - `drmaa_run_job`. Submit a job with attributes defined in the job template
  - `drmaa_run_bulk_jobs`. The *bulk* jobs are defined as a group of n similar jobs with a separate job id.
- Job control and monitoring routines: These routines are used to control and synchronize jobs, and monitor their status.

- `drmaa_control`. Start, stop, restart, or kill a job.
- `drmaa_synchronize`. Wait until all jobs specified, have finished execution.
- `drmaa_wait`. This routine waits for a single job to finish execution.
- `drmaa_job_ps`. Get the program status of a job.

Table 1 shows the implementation of the grid-oriented genetic algorithm describe above, using the DRMAA standard.

4 Experimental Results

In this section we will evaluate the functionality and efficiency of the Grid-oriented Genetic Algorithm described in Section 3, in the solution of the One-Max problem [10]. The One-Max is a classical benchmark problem for genetic algorithm computations, and it tries to evolve an initial matrix of zeros in a matrix of ones.

In our case we will consider an initial population of 1000 individuals, each one a 20x100 zero matrix. The sequential GA executed on each node will performed a fixed number of iterations (50), with a mutation and crossover probabilities of 0,1% and 60%, respectively. The exchange probability of best individuals between demes is 10%.

The following experiments were conducted on a research testbed made up of three different organizations, and based on the Globus Toolkit 2.4 [11]. See table 2 for a brief description of the resources in the testbed.

Table 2. Characteristics of the machines in the research testbed

Name	VO	Model	Speed	OS	Memory	DRMS
hydrus	UCM	Intel P4	2.5GHz	Linux 2.4	512MB	fork
cygnus	UCM	Intel P4	2.5GHz	Linux 2.4	512MB	fork
aquila	UCM	Intel PIII	700MHz	Linux 2.4	128MB	fork
babieca	CAB	5×Alpha DS10	450MHz	Linux 2.2	256MB	pbs

Figure 2 shows the execution profile of 4 generations of the GOGA, with a 5-way *dynamic connectivity*. Each subpopulation has been traced, and labeled with a different number (P_{deme}). As can be shown, individuals are exchanged between subpopulations $P1, P2, P3, P4, P5$ in the first generation; while in the third one the subpopulations used are $P1, P2, P4, P7, P8$. In this way the *dynamic connectivity*, introduces another degree of randomness since the demes that communicate differ each iteration and depend on the dynamism of the Grid.

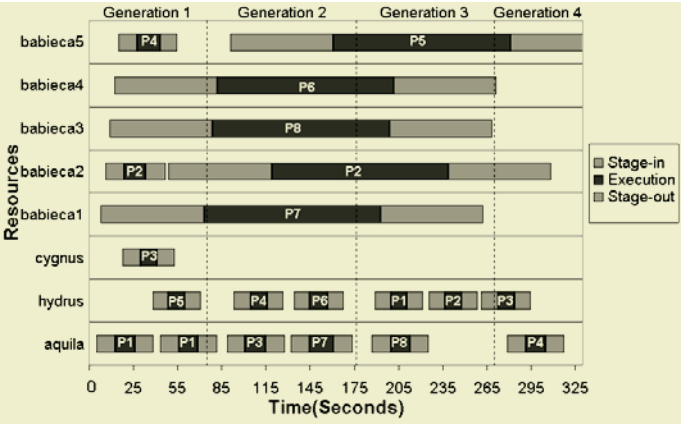


Fig. 2. Execution profile of four generations of the GOGA, each subpopulation has been labeled with P_{deme}

In order to study the effect of the dynamic connectivity we will consider five different executions, with different degrees of dynamic connectivity. In general, a high degree implies more demes exchanging individuals in each iteration, but also a higher execution time per iteration. On the other hand, a low degree reduces the iteration time, but deteriorates the numerical properties of the algorithm (the migration rate of individuals between subpopulations is also reduced). This effect is clearly shown in figure 3, in this case the optimum configuration is a 5-way connectivity.

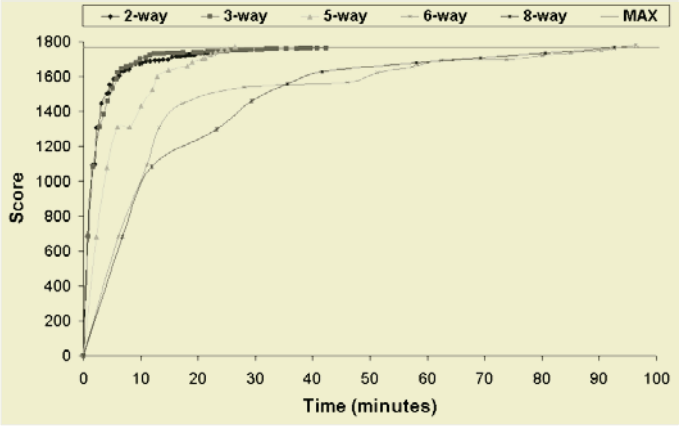


Fig. 3. Score versus execution time for five different degrees of connectivity

5 Conclusion

In this work we have presented an efficient Grid-oriented genetic algorithm. Our approach uses a fully connected multi-deme GA, with a dynamic connectivity between subpopulations to deal with the heterogeneity of the Grid. The optimum degree of connectivity depends on both, the computational characteristics of the Grid nodes, and the computational problem.

The GOGA has been developed taking advantage of the Grid*Way* framework features and the DRMAA API. In this way, it have been shown that DRMAA can aid the rapid development and distribution across the Grid of typical genetic algorithm strategies.

References

1. Kang, L., Chen, Y.: Parallel Evolutionary Algorithms and Applications. (1999)
2. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. J. of Software – Practice and Experience **34** (2004) 631–651
3. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report GFD-I.4, Scheduling Working Group – The Global Grid Forum (2001)
4. Huedo, E., Montero, R.S., Llorente, I.M.: Adaptive Scheduling and Execution on Computational Grids. J. of Supercomputing (2004) (in press).
5. Rajic, H., Brobst, R., Chan, W., Ferstl, F., Gardiner, J.: Distributed Resource Management Application API Specification 1.0. (2004)
6. Cant-Paz, E.: A Survey of Parallel Genetic Algorthms (1999)
7. Alba, E., Nebro, A.J., Troya, J.M.: Heterogeneous Computing and Parallel Genetic Algorithms. (2002)
8. Imade, H., Morishita, R., Ono, I., Ono, N., Okamoto, M.: A Grid-oriented Genetic Algorithm Framework for Bioinformatics. New Generation Computing **22** (2004) 177–186
9. Haas, A., Brobst, R., Geib, N., Rajic, H., Tollefsrud, J.: Distributed Resource Management Application API C Bindings v0.95. (2004)
10. Schaffer, J., Eshelman, L.: On Crossover as an Evolutionary Viable Strategy. In Belew, R., Booker, L., eds.: Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann (1991) 61–68
11. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications **11** (1997) 115–128

A Probabilistic Approach for Task and Result Certification of Large-Scale Distributed Applications in Hostile Environments*

Axel Krings, Jean-Louis Roch, Samir Jafar, and Sébastien Varrette

Laboratoire ID-IMAG (CNRS-INPG-INRIA-UJF – UMR 5132), Grenoble, France
{axel.krings, jean-louis.roch, samir.jafar, sebastien.varrette}@imag.fr

Abstract. This paper presents a new approach for certifying the correctness of program executions in hostile environments, where tasks or their results have been corrupted due to benign or malicious act. Extending previous results in the restricted context of independent tasks, we introduce a *probabilistic certification* that establishes whether the results of computations are correct. This probabilistic approach does not make any assumptions about the attack and certification errors are only due to unlucky random choices. Bounds associated with certification are provided for general graphs and for tasks with out-tree dependencies found in a medical image analysis application that motivated the research.

1 Introduction

Large scale global computing systems like the GRID and Peer-to-peer computing platforms gather thousands of resources for computing parallel applications, utilizing middleware infrastructures such as the Open Grid Service Architecture (OGSA) [2] to provide strong authentication, secure communications [11], and resource management. In this unbounded environment one should consider possible malicious act that may result in *massive attacks* against the whole global computation. This is supported by an exponentially increasing number of reported incidents [1], e.g. CERT/CC recorded close to 140,000 incidents in 2003.

Usually, global computations are expected to tolerate certain rates of faults [5, 9], e.g. small number of isolated intrusions. However, in order to ensure correctness of the computed results, one should detect if the global computation has been the victim of a massive attack resulting in an error rate larger than can be tolerated by the application.

The problem of protecting a computation against massive attacks has been mainly addressed for independent tasks. The analysis of *voting*, *spot-checking* and *credibility-based fault tolerance* is presented in [9]. An approach based on re-execution of tasks on reliable nodes is considered in [5], assuming that the

* This work has been supported by CNRS ACI Grid-DOCG and the Region Rhône-Alpes (Ragtime project).

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 323–333, 2005.

majority of workers are honest while workers compromised by an attack will always falsify their results. Under the same assumption, task dependencies are considered in [6], however, dependencies are used only for correction. Faults in systems with task dependencies are addressed in [4] where tasks are determined to execute on reliable or non-reliable nodes in order to maximize the expected number of correct results. Whereas the approach considers the critical issue of fault propagation, it is deterministic and therefore could be exploited by an intelligent adversary.

In order to eliminate any assumption on the attack and the distribution of errors in the context of a general parallel computation with dependencies, we propose to adopt a view directly inspired by probabilistic algorithms. Specifically, given the results of a global computation with task dependencies, we attempt to detect if the execution contains faulty results. Probabilistic algorithms are presented that make random choices and determine whether the execution is correct or faulty. Since the detection is probabilistic, its output may be wrong. However, contrary to previous approaches, the probability of certification error is not related to the application, i.e. the global computation, but only to the unlucky random choices associated with task selection for verification.

This work is motivated by medical applications [7] studied in the context of the French research project *Ragtime* [8] where certain highly computational manipulations of medical 3D/4D images, distributed over their production sites, allow for an acceptable fraction of error. The probabilistic certification algorithms presented can detect if these computations have been subjected to a massive attack with a so-called *attack ratio* greater than or equal to $q < 1$, with no other assumption about the attack. We show that pathological cases exist that minimize the probability of detection. The bound on the error is not related to q, but to the minimum number of so-called *initiator* tasks.

2 Definitions and Assumptions

Applications are executed on the global computing platform presented in [6]. A *user* initiates a computation, represented by a directed acyclic graph G, which is then executed on (a potentially large number of) unreliable *workers*. In order to verify the correctness of the results of the execution, *verifiers*, implemented by reliable resources which know graph G, re-execute selected tasks. Communication between workers and verifiers is through a checkpoint server containing computations submitted by workers [6]. Whereas any attack can occur on the worker or between the worker and the checkpoint server, the checkpoint server and verifiers are considered secure.

Dataflow Graph: The data-flow graph referred to above is a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a finite set of vertices v_j and $\mathcal{E}$ is a set of edges e_{jk}, $j \neq k$, representing precedence relations between $v_j, v_k \in \mathcal{V}$. The vertex set consists of two kinds of tasks. Let T_j denote the tasks as seen in the traditional context of task scheduling, i.e. a task is the smallest program unit of an instance

of execution. Let D_k denote a data task. Data tasks represent the inputs and outputs of a task. In the remainder of this paper, when talking about a task, it is implied to be a task T_j. Data tasks will be referred to as inputs or outputs of T_j. The total number of tasks T_j in G is n.

Executions and the Impact of Faults: We will first establish the notion of program execution and the impact of faults. Let E denote the execution of a workload represented by G with a set $\hat{I}$ of initial inputs on a set of unreliable resources. It is assumed that G is static, i.e. it is fixed. Each task T in E executes with inputs $i(T, E)$ and creates output $o(T, E)$. The inputs of a task T_j are composed of either inputs from $\hat{I}$ or outputs of other tasks T_k, i.e. $o(T_k, E)$.

Let $\hat{E}$ denote the execution of the program on a verifier, i.e. a reliable resource, or set thereof. If $E = \hat{E}$, i.e. if every task in E uses the same inputs and computes the same outputs as those in $\hat{E}$, then E is said to be "correct". Conversely, if $E \neq \hat{E}$, then at least one task in E produced a wrong result and the execution is said to have "failed".

In order to differentiate whether a task execution is considered to be on a client or verifier and whether the inputs and outputs of the execution are those of E or $\hat{E}$, the following notation is adapted. Note that a "hat" always refers to a reliable resource, input or output. Let $i(T, E)$ denote the input of T in E and $\widehat{i}(T, \hat{E})$ the input of T in $\hat{E}$. Furthermore, let $o(T, E)$ denote the output of T on the client, $\hat{o}(T, E)$ the output of T on the verifier based on inputs from E, and $\hat{o}(T, \hat{E})$ the output of T on the verifier based on inputs from $\hat{E}$. Note that the notations $\hat{o}(T, E)$ and $\hat{o}(T, \hat{E})$ differ. Both indicate outputs generated on a verifier, but the first assumes $i(T, E)$ and the latter $\widehat{i}(T, \hat{E})$ as inputs.

Probabilistic Certification: We consider probabilistic certification based on a probabilistic algorithm that uses randomization in order to state if E has failed or not. Given an execution E, a Monte Carlo certification is defined as a randomized algorithm that takes an arbitrary ϵ, $0 < \epsilon \leq 1$, as input and delivers (1) either *CORRECT* or (2) *FAILED*, together with a proof that E has failed. The probabilistic certification is said to be with error ϵ if the probability of the answer *CORRECT*, when E has actually failed, is less than or equal to ϵ.

For instance, a Monte Carlo certification may consist of re-executing randomly chosen tasks in G on a verifier, comparing results to those obtained in E. If the results differ E has failed. Otherwise, E may be correct or failed. However, if E has failed, a probabilistic certification with error ϵ ensures that the probability of non-detection of failure (based on randomly selecting tasks in G for re-execution) is less than or equal to ϵ.

Monte Carlo Certification Against Massive Attacks: In the sequel we denote the number of forged tasks in G by n_F. We are considering the two scenarios where either all tasks execute correctly, i.e. $n_F = 0$, or n_F is large, corresponding to a massive attack. A *massive attack* with *attack ratio* q consists of falsifying the execution of at least $n_q = \lceil qn \rceil \leq n_F$ tasks. E is said to be "attacked with ratio q" and $\frac{n_F}{n} \geq q$. It should be noted that q is assumed relatively large, see

Section 5, resulting from massive attacks such as caused by a virus or Trojan. The objective is to provide a probabilistic Monte Carlo certification against such massive attacks. Note that detection of small attacks, e.g. single intrusions, is not the scope of this work. As indicated in Section 1, global computations are expected to tolerate certain fault rates.

3 Certification of Independent Tasks

We first consider the case where all tasks in G are independent. In this case, certification of tasks is equivalent to certification of results. The following Monte-Carlo Test (MCT), based on task re-execution on a verifier, will be used to detect if execution E contains forged tasks.

Algorithm MTC

1. Uniformly choose one task T in G. The input and output of T in E are $i(T, E)$ and $o(T, E)$ respectively.
2. Re-execute T on a verifier, using inputs from E, i.e. $i(T, E)$, to get output $\hat{o}(T, E)$. If $o(T, E) \neq \hat{o}(T, E)$ return FAILED;
3. Return CORRECT.

Since all tasks in G are independent[1] we always have $i(T, E) = \widehat{i}(T, \hat{E})$. If Algorithm MTC selects a forged task, then one knows with certainty that the execution E has failed. However, if MTC returns CORRECT, then one can only make conclusions based on the probabilities of randomly selecting a falsified or non-falsified task. The following lemma addresses these probabilities.

Lemma 1. *Let E be an execution with n independent tasks, n_F of which have been forged. The probability that MTC returns FAILED is $\frac{n_F}{n}$ and the probability that it returns CORRECT is $1 - \frac{n_F}{n} \leq 1 - q$.*

Proof. The probability that MTC chooses a forged task is $\frac{n_F}{n}$. Then the probability that MTC returns CORRECT is $\frac{n-n_F}{n} = 1 - \frac{n_F}{n} \leq 1 - q$. □

The theorem below gives a lower bound on the number of tasks to be re-executed in order to achieve a specific ϵ.

Theorem 1. *Let E be an execution with only independent tasks and assume that E is either correct or massively attacked with ratio q. For a given ϵ, the number of independent executions of algorithm MTC necessary to achieve a certification of E with probability of error less than or equal to ϵ is $N \geq \lceil \frac{\log \epsilon}{\log(1-q)} \rceil$.*

Proof. Consider N executions of Algorithm MTC. If during any of the N executions MTC selects a forged task, the execution has failed. Therefore, assume

[1] In the case of task dependencies this assumption about the inputs does not hold anymore, as will be addressed later.

that only non-forged tasks are selected. According to Lemma 1 the probability of MTC selecting a non-forged task is $\frac{n-n_F}{n} \leq 1-q$. Then N independent applications of MTC lead to a Monte-Carlo certification with a probability of error bound by $\epsilon \leq (1-q)^N$. For a given ϵ, it is thus sufficient to select $N \geq \lceil \frac{\log \epsilon}{\log(1-q)} \rceil$ tasks. □

4 Certification in the Presence of Task Dependencies

In the previous section there is no difference between certification of tasks and their respective results. If one allows for dependencies among tasks the certification of the results of tasks is more difficult. The problem lies in the way a reliable resource has to determine the validity of results. Any measure of validity of a task's result based on the comparison to the results obtained by re-executing the same task on a reliable resource, depends on the validity of the inputs the reliable resource uses for re-execution. Just the fact that the outputs of a task execution and its re-execution on a reliable resource produce identical results does not say much about the validity of that result, since in the assumed deterministic computing environment the same faulty input will produce identical faulty output. Thus, in the presence of dependencies, $o(T, E) = \hat{o}(T, E)$ only indicates that the results are the same, but not that they are correct. It should be noted that correctness would imply that $o(T, E) = \hat{o}(T, \hat{E})$.

4.1 Faulty Tasks and the Concept of Initiators

The randomized testing used in Section 3 is only valid for result certification of independent tasks. If we were to apply the same reasoning in the presence of dependencies, certification based on repeated application of Algorithm MTC would only certify results if $o(T, E) \neq \hat{o}(T, E)$ for each falsified T selected by MTC. However, this assumption is too restrictive since it would assume that a re-execution with some (perhaps incorrect) input values would always expose[2] the forgery. This weak assumption could be easily exploited by an attacker.

Suppose Algorithm MTC is used. If $o(T, E) \neq \hat{o}(T, E)$ then E has failed. However, $o(T, E) = \hat{o}(T, E)$ indicates a correct output only if the inputs are correct, i.e. $\widehat{i}(T, \hat{E})$. This implies that T has no forged predecessors. In the following discussion, falsified tasks which have no falsified predecessors will be called *initiators*. The probabilities associated with randomly selecting initiators will be the basis for result certification. It should be noted that it is difficult to speculate on the capabilities of detecting incorrect results of falsified tasks that are not initiators. Pathological attacks may be derived where the output of one falsified task may be custom tailored to produce results for other falsified tasks that do not differ from their re-executions (with the forged inputs) on reliable resources.

[2] It should be noted that the task inputs define a rather limited "test vector" for the task. The quality of test vectors with respect to fault coverage has been extensively studied in the context of the *Test Vector Generation Problem* [3].

4.2 Certification

Result certification is directly related to the probability of the certification algorithm selecting initiators. Let n_I denote the number of initiators in G. Note that the determination of n_I depends on the graph and which nodes have been falsified. The following lemma and theorem, modified from Lemma 1 and Theorem 1, can be stated.

Lemma 2. *Let E be an execution with n tasks with dependencies. Furthermore, let n_F and n_I be the number of forged tasks and initiators respectively, $n_I \leq n_F$. The probability that MTC returns FAILED is at least $\frac{n_I}{n}$ and the probability that it returns CORRECT is less than or equal to $1 - \frac{n_I}{n}$.*

Proof. The probability that MTC selects an initiator, and thus returns value FAILED, is $\frac{n_I}{n}$. Then the probability that MTC returns CORRECT is less than or equal to $\frac{n-n_I}{n} = 1 - \frac{n_I}{n}$. □

Lemma 3. *Let E be an execution of tasks with dependencies and assume that E is either correct or massively attacked with ratio q. For a given ϵ, the number of independent executions of algorithm MTC necessary to achieve a certification of E with probability of error less than or equal to ϵ is $N \geq \lceil \frac{\log \epsilon}{\log(1-\frac{n_I}{n})} \rceil$.*

Proof. Consider N executions of Algorithm MTC. If during any of the N executions MTC selects an initiator, the execution has failed. Therefore, assume that only non-initiator tasks are selected. According to Lemma 2 the probability of MTC selecting a non-initiator task is $\frac{n-n_I}{n} = 1 - \frac{n_I}{n}$. Then N independent applications of MTC lead to a Monte-Carlo certification with a probability of error bound by $\epsilon \leq (1 - \frac{n_I}{n})^N$. For a given ϵ, it is thus sufficient to select $N \geq \lceil \frac{\log \epsilon}{\log(1-\frac{n_I}{n})} \rceil$ tasks. □

Unlike the case of Theorem 1, this result is more restrictive since the value of n_I depends on G under consideration of the worst-case attack scenario and it is likely to be small. In the remainder of the paper we study the implications of worse case attacks, where the attacker knows or can estimate the structure of the graph. We will prove that even in pathological cases a lower bound on the number of initiators n_I can be defined, considering G and attack ratio q.

Let $G^<(T)$ denote the sub-graph induced by all predecessors of a task T or a set of tasks V, i.e. $G^<(V)$. Furthermore, let $G^\leq(T) = G^<(T) \cup \{T\}$. The graphs of successors are denoted similarly, i.e. $G^>(T)$ and $G^\geq(T)$. We now formally define the set of initiators.

For a given G with n tasks let F denote the set of all falsified tasks. The *initiator set* $\mathcal{I}(F)$ is defined as the set of all $T_i \in F$ which have no predecessors in F, i.e. $\mathcal{I}(F) = \{T_i \in F : F \cap G^<(T_i) = \emptyset\}$. It is obvious that the actual tasks in sets F and $\mathcal{I}(F)$ are not known, since otherwise certification would be trivial.

Since re-execution of a task with incorrect inputs may still result in $o(T, E) = \hat{o}(T, E)$ one has to consider the limitations induced by the inputs.

Lemma 4. *Given the set of all falsified tasks F and an arbitrary T in G, if the outputs of T are not correct, then it must be that $G^{\leq}(T) \cap \mathcal{I}(F) \neq \emptyset$.*

The proof follows directly from the fact that, if the output of T is not correct, then either T is faulty, i.e. $T \in F$, or there must be at least one forged task in predecessor set $G^{<}(T)$. Determining that the output of T is incorrect may require to verify few or many tasks and is largely dictated by the size of $G^{\leq}(T)$. The following definitions will aid in capturing the difficulties associated with determining whether results are incorrect, considering that a pathological attacker will minimize the probability of finding forged tasks.

First, the minimum number of initiators with respect to given subgraph of G is defined. Let V be a set of tasks in G and let $k \leq n_F$ be the number of falsified tasks assumed. Define $\gamma_V(k)$ as the *minimum number of initiators* with respect to V and k such that $\gamma_V(k) = min|G^{\leq}(V) \cap \mathcal{I}(F)|$ for $|F| \geq k$ and of all $G^{\leq}(V) \cap \mathcal{I}(F) \neq \emptyset$. Recall that $n_q \leq n_F$ is the smallest number of falsified tasks as the result of an attack with ratio q. Then $\gamma_G(n_q)$ is the smallest number of initiators possible, e.g. the number associated with a pathological attack scenario. With respect to Lemma 2 the probability that MTC returns correct can thus be written as $1 - \frac{\gamma_G(n_q)}{n}$.

Next, we will define the *minimal initiator ratio* $\Gamma_V(k)$ as

$$\Gamma_V(k) = \frac{\gamma_V(k)}{|G^{\leq}(V)|}. \tag{1}$$

The minimum initiator ratio is helpful in determining bounds on probabilities of selecting initiators in predecessor sets. This is of interest when the structure of G will allow for adjustments of probabilities of finding initiators based on the specific task considered by an algorithm like MTC. With respect to G this allows the number of verifications in Lemma 3 to be expressed as $N \geq \lceil \frac{\log \epsilon}{\log(1-\Gamma_G(n_q))} \rceil$.

4.3 The Impact of Graph G

Knowing the graph, an attacker may attempt to minimize the visibility of even a massive attack with ratio q. From an attacker's point of view, it is advantageous to "hide" falsified nodes in the successor graphs of certain tasks in order to achieve the pathological minimum number of initiators $\gamma_G(n_F)$. This allows to generate a general bound on $\gamma_G(n_F)$ based on the the size of successor graphs.

Lemma 5. *Given height h (the length of the critical path) and maximum out-degree d of a graph G, the minimum number of initiators is*

$$\gamma_G(n_F) = \lceil \frac{n_F}{\left(\frac{1-d^h}{1-d}\right)} \rceil. \tag{2}$$

Proof. Given h and d and any task T_i in G the maximal size of the successor graph $G^{\geq}(T_i)$ is bound by $|G^{\geq}(T_i)| \leq 1 + d + d^2 + \ldots + d^{h-1} = \frac{1-d^h}{1-d}$. Thus, a single initiator T_i can "hide" at most $\frac{1-d^h}{1-d}$ of the n_F falsified tasks in F. This

would, by definition, make all tasks in $G^{>}(T_i)$ non-initiators. If each initiator can "hide" a maximum of $\frac{1-d^h}{1-d}$ tasks in F, the minimum number of such initiators is $\lceil \frac{n_F}{\left(\frac{1-d^h}{1-d}\right)} \rceil$. Note that this term is has the smallest value for $n_F = n_q$. □

For specific graphs this general worst-case scenario may be overly conservative. The following Extended Monte-Carlo Test (EMCT) will allow graphs with relatively small predecessor subgraphs to overcome the restrictions imposed by $\gamma_G(n_q)$. Note that it is similar to Algorithm *MTC* except that it contains previsions to verify all predecessors for the task T selected for verification. Thus, it effectively verifies $G^{\leq}(T)$.

Algorithm EMTC

1. Uniformly chose one task T in G.
2. Re-execute all T_j in $G^{\leq}(T)$, which have not been verified yet, with input $i(T_j, E)$ on a reliable resource and return FAILED if for any T_j we have $\hat{o}(T_j, E) \neq o(T_j, E)$.
3. Return CORRECT.

In the context of our application domain in which G consists mainly of out-trees, Algorithm *EMTC* exhibits its strength, e.g. the worst number of re-executions in Step 2 is less than height h.

Theorem 2. *For a single execution of Algorithms EMTC the probability of error is $e_E \leq 1 - q$. The average cost in terms of verification, i.e. the expected number of verifications, is*

$$C = \frac{\sum_{T_i \in G} |G^{\leq}(T_i)|}{n}. \tag{3}$$

Proof. A pathological attacker who knows that uniform random task selection is used and that all predecessor tasks are verified can minimize detection by falsifying tasks in such a way as to minimize error propagation, thereby minimizing the total number of tasks affected by falsifications. In other words, in the worst case n_q falsified tasks in G are distributed so that the number of T whose $G^{\leq}(T)$ contain falsified tasks is minimized. This can be achieved in any scenario which attacks the n_q tasks T_i with the smallest successor graph $G^{\geq}(i)$, e.g. first attack only leaf tasks, then tasks at the second level etc. until n_q tasks have been attacked. Finally, the error e_E is 1 minus the probability of $G^{\leq}(T)$ containing a faulty task. In the worst case described above this leads to $e_E \leq 1 - \frac{n_q}{n} \leq 1 - q$.

The average number of verifications is simply the average number of tasks in the predecessor graph verified in *EMTC* Step 2. Note that once T is selected, the cost can be specified exactly as $|G^{\leq}(T)|$. □

Table 1. Results for general graph and forest of out-trees

Algorithm	*MTC*	*EMTC*
Number of effective initiators	$\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil$	n_q
Probability of error	$1 - \frac{\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil}{n}$	$1-q$
Verification cost: general G	1	$O(n)$
Verification cost: G is out-tree	1	$h - log_d(n_v)$
Ave. # effective initiators, G is out-tree	$\lceil \frac{n_q}{\left(\frac{1-(h+2)d^{h+1}+(h+1)d^{h+2}}{(1-d)(1-d^{h+1})}\right)} \rceil$	n_q

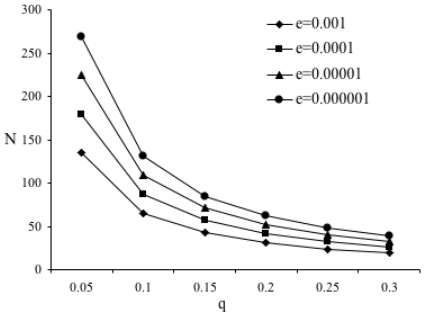


Fig. 1. Impact of q on N

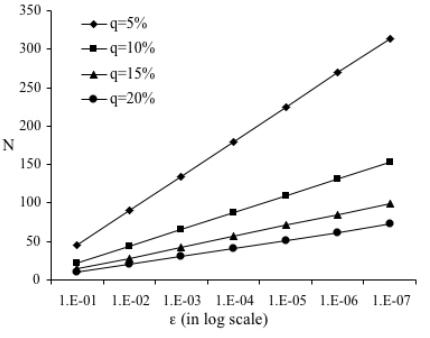


Fig. 2. Impact of ϵ on N

5 Results

Table 1 shows results with respect to a single invocation of the algorithm specified for pathological cases associated with general graphs and out-trees (as indicated). The number of effective initiators is the number of initiators as perceived by the algorithm[3]. The probability of error is a direct result of the number of effective initiators. The cost of verification reflects the number of tasks verified for each invocation of the algorithm. When G is a forest of out-trees the cost of verification changes based on the number of tasks verified, n_v, since only non-verified tasks are re-executed. If attacks are random, the average number of effective initiators depends on the average size of successor graphs.

The impact of ϵ and q on the number of invocations of *MTC* or *EMTC* is shown next. The results shown in the Figures 3 and 3 are identical for certification of independent tasks with *MTC* and dependent tasks with *EMTC*. For different values of ϵ Figure 3 shows the impact of attack ratio q on N. It should be noted that N decreases fast with increasing values of q. Conversely, it shows that

[3] The term "effective" initiator is used to emphasize that in Algorithm *EMTC* step 2 any falsification in $G^{\leq}(T)$ is guaranteed to result in detection.

probabilistic detection is unsuitable for very small q. Next, Figure 3 shows the impact of ϵ on N for fixed attack ratios q. Note that N only grows logarithmically in ϵ. This is very desirable, as it allows for certification with high degrees of certainty, i.e. very small ϵ.

6 Conclusion

This paper discussed certification of large distributed applications executing in hostile environments, where tasks or data may be manipulated by attacks. Unlike previous work based on independent tasks, we considered fault propagation occurring in applications with dependent tasks. In addition we used a probabilistic approach with no assumptions about fault behavior. Two probabilistic algorithms were introduced that selected a small number of tasks to be re-executed on a reliable resource, indicating correct execution with a probability of error based on probabilities associated with task selection. Task re-execution was based on utilizing inputs available through macro data-flow checkpointing. By introducing the concept of initiators, the fault-detection problem associated with fault propagation were overcome. The cost for this were additional verifications noticeable in Algorithm $EMTC$. However, in the context of real-world applications, represented by out-trees, this translated to only minor overhead.

References

1. CERT/CC Statistics 1988-2004, CERT Coordination Center, http://www.cert.org/stats/cert_stats.html
2. Foster, I., Kesselman, C., Nick, J. and Tuecke, S., *Grid Services for Distributed System Integration*, IEEE Computer, No. 6, Vol. 35 (2002) 37-46
3. Fujiwara Hideo, *Logic Testing and Design for Testability*, MIT Press, 1985
4. Gao, L., and Malewicz, G., *Internet Computing of Tasks with Dependencies using Unreliable Workers*, 8th International Conference on Principles of Distributed Systems (OPODIS'04), Dec., 15-17, 2004 (to appear)
5. Germain, C., and Playez, N., *Result Checking in Global Computing Systems*, Proceedings of the 17th Annual ACM International Conference on Supercomputing (ICS 03), San Francisco, California, 23-26 June, (2003) 218-227
6. Jafar S., Varrette S., and Roch J.-L., *Using Data-Flow Analysis for Resilence and Result Checking in Peer to Peer Computations*, Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA 2004), Zaragoza, Espagne, 30^{th} Aug. - 3^{rd} Sep., (2004) 512-516
7. Montagnat, J., Breton, V., and Magnin, I., *Partitioning medical image databases for content-based queries on grid*, Methods of Information in Medicine, Special Issue HealthGrid04, 2004, (to appear)
8. Ragtime: Grille pour le Traitement d'Informations Médicales, Région Rhône-Alpes http://liris.univ-lyon2.fr/∼miguet/ragtime/

9. Sarmenta, Luis F.G., *Sabotage-Tolerance Mechanisms for Volunteer Computing Systems*, Future Generation Computer Systems, Vol. 18, Issue 4 (2002) 561-572
10. Wasserman, H., and M. Blum, Software reliability via run-time result-checking, Journal of the ACM, Vol. 44, No. 6 (1997) 826-849
11. Von Welch, et.al., *Security for Grid Services*, 12^{th} Intl. Symposium on High Performance Distributed Computing (HPDC-12), 22-24 June, Seattle, WA, (2003) 48–57

A Service Oriented Architecture for Decision Making in Engineering Design

Alex Shenfield and Peter J. Fleming

Department of Automatic Control and Systems Engineering,
University of Sheffield,
Mappin Street, Sheffield, S1 3JD, United Kingdom
A.Shenfield@sheffield.ac.uk

Abstract. Decision making in engineering design can be effectively addressed by using genetic algorithms to solve multi-objective problems. These multi-objective genetic algorithms (MOGAs) are well suited to implementation in a Service Oriented Architecture. Often the evaluation process of the MOGA is compute-intensive due to the use of a complex computer model to represent the real-world system. The emerging paradigm of Grid Computing offers a potential solution to the compute-intensive nature of this objective function evaluation, by allowing access to large amounts of compute resources in a distributed manner. This paper presents a grid-enabled framework for multi-objective optimisation using genetic algorithms (MOGA-G) to aid decision making in engineering design.

1 Introduction

Soft Computing techniques such as Neural Networks, Fuzzy Logic, and Evolutionary Computation are used to solve many complex real-world engineering problems. These techniques provide the engineer with a new set of tools that often out-perform conventional methods in areas where the problem domain is noisy or ill-defined. However, in the cases of Neural Networks and Evolutionary Computation especially, these tools can be computationally intensive.

Grid Computing offers a solution to the computationally intensive nature of these techniques. The Grid Computing paradigm is an emerging field of computer science that aims to offer "a seamless, integrated computational and collaborative environment" [1]. Ian Foster defines a computational grid as "a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [2]. Grid Computing is differentiated from conventional distributed computing by its emphasis on coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations [3]. These resources include software packages, compute resources, sensor arrays, data and many others.

The purpose of this paper is to introduce a grid enabled framework for multi-objective optimisation using genetic algorithms (MOGA-G). This framework will

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 334–343, 2005.

be presented in the context of a Service Oriented Architecture approach. This approach ties in with that taken by the Globus Project [4] to providing access to grid resources via Grid Services. Section 2 will introduce Genetic Algorithms and Multi-Objective Optimisation. Section 3 will briefly introduce the core grid concepts used in the implementation of our framework. Section 4 will outline other related work. Section 5 will provide details of the implementation of our framework, and Section 6 will draw some conclusions and present some ideas for further work.

2 Genetic Algorithms and Multi-objective Optimisation

2.1 Genetic Algorithms

Genetic Algorithms (GAs) are an optimisation technique utilising some of the mechanisms of natural selection [5]. GAs are an iterative, population based method of optimisation that are capable of both exploring the solution space of the problem and exploiting previous generations of solutions. Exploitation of the previous generation of solutions is performed by a selection operator. This operator gives preference to those solutions which have high fitness when creating the next generation of solutions to be evaluated. Exploration of the solution space is performed by a mutation operator and a recombination operator and helps to ensure the robustness of the algorithm by preventing the algorithm from getting stuck in local optima.

Genetic Algorithms evaluate candidate solutions based on pay-off information from the objective function, rather than derivative information or auxiliary knowledge. This ensures that GAs are applicable to many different problem domains, including those where conventional optimisation techniques (such as hill-climbing) may fail.

2.2 Multi-objective Optimisation

Many real-world engineering design problems involve the satisfaction of multiple conflicting objectives. In this case it is unlikely that a single ideal solution will be possible. Instead, the solution of a multi-objective optimisation problem will lead to a family of Pareto optimal points, where any improvement in one objective will result in the degradation of one or more of the other objectives.

Genetic Algorithms are particularly well suited to this kind of multi-objective optimisation, because they search a population of candidate solutions. This enables the GA to find multiple solutions which form the Pareto optimal set (see Fig. 1). GAs are often able to find superior solutions to real-world problems than conventional optimisation techniques (i.e. constraint satisfaction). This is due to the difficulty that conventional optimisation techniques have when searching in the noisy or discontinuous solution spaces that real-world problems often have.

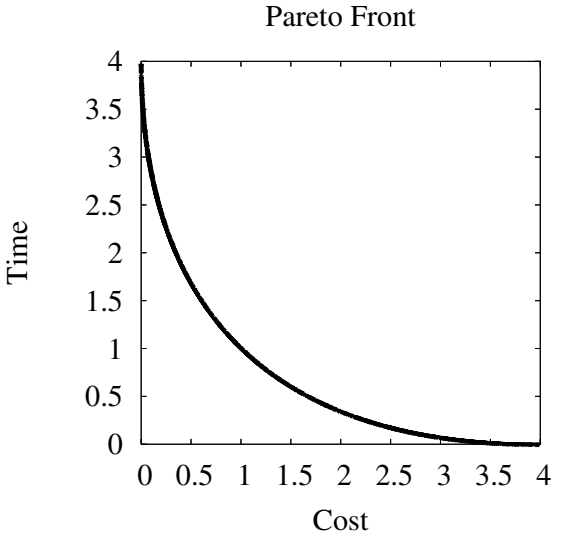


Fig. 1. The Pareto Optimal Solution Set

2.3 Applications of Genetic Algorithms

Genetic Algorithms have been used to solve problems across many different disciplines. GAs have been used in such diverse fields as Economics and Social Theory [6], Robotics [7] and Art [8]. For many non-trivial real-world applications the evaluation of the objective function is performed by computer simulation of the system. For example, in the optimisation of controller parameters for gas turbine aero engines [9], a computer model of the engine is used to calculate the values of the objective functions for a given controller design.

The use of computer simulations to evaluate the objective function leads to some new issues. To ensure that the results gained from the genetic algorithm are meaningful, the simulation must be complex enough to capture all the relevant dynamics of the true system. However, assuming that this level of complexity is obtainable, the simulation may be very computationally intensive. As genetic algorithms are population based methods, the simulation must be run many times. In a typical genetic algorithm this could involve running the simulation 10,000 times.

2.4 Parallel Genetic Algorithms

The computationally intensive nature of the evaluation process has motivated the development of parallel genetic algorithms. Early proposals for the implementation of parallel GAs considered two forms of parallelisation which still apply today: multiple communicating populations, and single-population master-slave implementations [10].

The decision between which of these two types of parallelisation to implement must consider several factors, such as ease of implementation and use, and the performance gained by parallelisation. Single-population parallel GAs are often the easier to implement and use, as experience gained with sequential GAs can be easily applied to these. In contrast, the implementation and use of multiple communicating populations based parallel GAs involves choosing appropriate values for additional parameters such as size and number of populations, frequency of migration, and the number of individuals involved in migration. This increases the complexity of the parallel GA as each of these parameters affects the efficiency of the algorithm and the quality of the overall solution.

3 Grid Technologies

The concept of Grid Computing is not new. As far back as 1969 Len Kleinrock suggested:

> "We will probably see the spread of 'computer utilities', which, like present electric and telephone utilities, will serve individual homes and offices across the country." [11]

However, it is only recently that technologies such as the Globus Toolkit have emerged to enable this concept to be achieved. The Globus Toolkit is an open-source, community-based set of software tools to enable the aggregation of compute, data, and other resources to form computational grids. Since version 3 of the Globus Toolkit it has been based on the Open Grid Services Architecture (OGSA) introduced by the Globus Project. OGSA builds on current Web Service concepts and technologies to support the creation, maintenance, and application of ensembles of services maintained by virtual organisations [12].

3.1 Web Services

A Web Service is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages" [13]. Web Services are accessible through standards-based internet protocols such as HTTP and are enabled by three core technologies [14]:

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery, and Integration (UDDI)

These technologies work together in an application as shown in Fig. 2. The Web Service client queries a UDDI registry for the desired service. This can be done by service name, service category, or other identifier. Once this service has

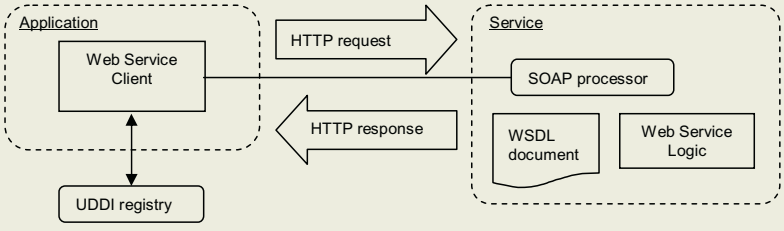


Fig. 2. Interaction between Web Service Technologies

been located the client queries the WSDL document to find out how to interact with the service. The communication between client and service is then carried out by sending and receiving SOAP messages that conform to the XML schema found in the WSDL document.

3.2 Open Grid Services Architecture

The Open Grid Services Architecture (OGSA) is the basis for the Globus Toolkit version 3. OGSA represents computational resources, data resources, programs, networks and databases as services. These services utilise the Web Services technologies mentioned in Section 3.1. There are three main advantages to representing these resources as services:

1. *It aids interoperability.* A service-oriented view addresses the need for standard service definition mechanisms, local/remote transparency, adaptation to local OS services, and uniform semantics [12].
2. *It simplifies virtualisation.* Virtualisation allows for consistent resource access across multiple heterogeneous platforms by using a common interface to hide multiple implementations [12].
3. *It enables incremental implementation of grid functionality.* The provision of grid functionality via services means that the application developer is free to pick and choose the services that provide the desired behaviour to their application.

4 Related Work

In recent years the interest in using parallel genetic algorithms to solve single-objective optimisation problems has increased considerably [15]. However, there has been little research performed in applying parallel GAs to solve multi-objective optimisation problems. In [16] and [17] there is some discussion concerning multi-objective evolutionary optimisation techniques in distributed systems, but these do not implement parallel GAs in a Grid Computing environment.

A middleware system for evolutionary computation in a Grid Computing environment is proposed in [18], and then used to construct a parallel simulated annealing algorithm to solve a single objective problem. This system requires the application developer to implement a set of interfaces (comprising the middleware) and write the code for the desired evolutionary operations. Another paper that utilizes the Grid Computing concept for single-objective optimisation using genetic algorithms is [19]. This paper develops a 'Black Box Optimisation Framework' (BBOF) in C++ to optimise a computer simulation of a forest fire propagation problem from environmental science. This BBOF is executed in a Condor pool to harness the spare CPU cycles of a cluster of computers.

Our MOGA-G system differs from those proposed in [18] and [19] because it provides a concrete implementation of a *multi-objective genetic algorithm*. Like [19] we have utilised the power of computational grids to perform distributed fitness evaluation of our objectives, but we have implemented our framework in a Service Oriented Architecture using the Globus Toolkit to provide access to the resources of the grid (see section 5.2).

The power of computational grids is used to execute a distributed enumerative search in [20]. This distributed enumerative search is then used to generate the Pareto-optimal front for several benchmark test functions that are commonly used in the evaluation of the performance of multi-objective optimisation algorithms. A brief comparison with heuristic techniques is then performed.

This MOGA-G system is more computationally efficient than the distributed enumerative search described in [20]. This is because our algorithm converges on the Pareto optimal front by making intelligent choices about which points to search in the next generation, whereas the enumerative search algorithm has to evaluate every point in the search space. This approach would be impossible for a real-world engineering design problem due to the potential size of the search space.

5 Implementation

5.1 Parallelisation of the Multi-objective Genetic Algorithm

In section 2.4 we found that there are two types of possible parallelisation strategies for genetic algorithms: multiple communicating populations, and single-population master-slave implementations. In the implementation of our grid-enabled framework for multi-objective optimisation using genetic algorithms (MOGA-G) we have decided to parallelise our multi-objective genetic algorithm using the single-population master-slave implementation. This is also known as distributed fitness evaluation or global parallelisation. This model uses the master-worker paradigm (see Fig. 3) of parallel programming.

A master-slave parallel genetic algorithm uses a single population maintained globally by the master node and parallelises the evaluation of the objective function by distributing the population to the worker processes. These are then assigned to the available processors for execution (in the ideal case, one individual

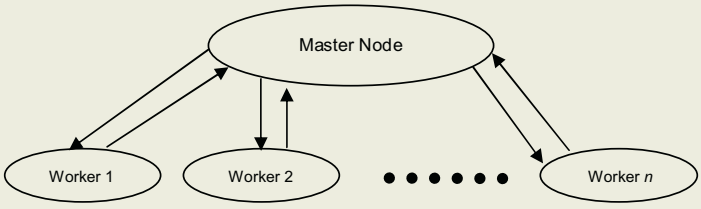


Fig. 3. The Master-Worker Programming Paradigm

per processor). The genetic operators - selection, recombination and mutation - are then applied globally by the master node to form the next generation.

This model is particularly well suited for the parallelisation of genetic algorithms as the evaluation of the objective function requires only the knowledge of the candidate solution to evaluate, and therefore there is no need for inter-communication between worker processes. Communication only occurs when the individuals are sent to the worker processes for evaluation and when the results of those evaluations are returned to the master node.

5.2 Service-Oriented Architecture and the Globus Toolkit Version 3

We have chosen to implement our grid-enabled framework for multi-objective optimisation using genetic algorithms in a Service-Oriented Architecture (SOA) using the Globus Toolkit version 3 to provide access to the resources of the grid. We have implemented the MOGA-G framework using the Java programming language, primarily due to the portability of the code. This means that the components of the MOGA-G framework can easily be run across various heterogeneous platforms.

A service-oriented architecture is essentially a collection of services that communicate with each other in order to perform a complex task. SOA is an approach to building loosely-coupled, distributed systems that combine services to provide functionality to an application. IBM sees SOA as key to interoperability and flexibility requirements for its vision of an on demand business [21].

The SOA approach to grid computing is well suited to the kind of master-worker parallelism used in the MOGA-G framework. This SOA view of grid computing has the client acting as the master node, and the service acting as the worker. In the implementation of the MOGA-G framework (see Fig. 4) there are two different services. One service exposes the operations of the multi-objective genetic algorithm to the client, and the other provides operations for running evaluations of the objective function on the computational grid.

This SOA approach also provides flexibility both in how the MOGA-G framework is used and in the maintenance of the framework. The provision of the components of the MOGA-G framework as services means that it is simple to

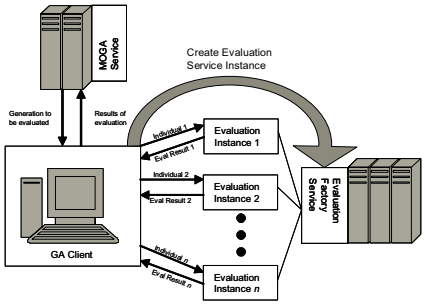


Fig. 4. The Implementation of the MOGA-G Framework

add new functionality to the system, and to improve upon existing functionality, by adding new services. In the context of the MOGA-G framework, this functionality could be anything from the implementation of the genetic algorithm operators - selection, recombination and mutation - to the distribution and management of the objective function evaluation.

Providing the MOGA-G framework as services also means that the functionality can be accessed via the HTTP protocol. This means that the services can be easily integrated into an Internet portal so as to be accessible by any device with a capable web browser (such as a PDA).

This SOA approach is used in providing access to grid resources via the Globus Toolkit (see section 3). The Globus Toolkit has become a fundamental enabling technology for grid computation, letting people carry out computations across geographically distributed resources in a secure way. The success of the Globus Project has meant that the project has become one of the driving forces in developing standards for grid computing.

6 Conclusions and Further Work

This paper has described a grid-enabled framework for multi-objective optimisation using genetic algorithms (MOGA-G). This MOGA-G framework has been designed in a Service-Oriented Architecture (SOA) so as to take advantage of the flexibility that this architecture offers. In the MOGA-G framework a concrete implementation of a multi-objective genetic algorithm is provided. However, the SOA approach that we have taken allows our implementation to be easily extended to provide additional features, such as those required to construct hybrid genetic algorithms. Extending the MOGA-G framework to support additional features is an area for further investigation.

This framework is primarily suited to computationally expensive objective function evaluations, such as those performed by computer simulation, due to its distributed nature. For computationally trivial objective functions the communication overheads involved in executing the evaluations in a distributed manner

result in a decrease in performance compared to a sequential GA. This is due to the way in which job submission and management is performed. Whilst further work will be conducted into determining the scale of problems for which this framework is most effective, it is expected that further research and development of grid-middleware, job submission services, and job management services will provide a reduction in these communication overheads. This will allow our framework to provide increased performance for less computationally intensive problems. However, this framework is not intended to replace sequential GAs in cases where the performance of the sequential GA is satisfactory.

References

1. Baker, M., Buyya, R., and Laforenza, D., Grids and Grid technologies for wide-area distributed computing, Software: Practice and Experience, **32(15)**, pp. 1437–1466, 2002.
2. Foster, I., and Kesselman, C. (eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999.
3. Foster, I., Kesselman, C., and Tuecke, S., The Anatomy of the Grid: Enabling Scalable Virtual Organizations, Int. J. Supercomputer Applications, **15(3)**, 2001.
4. The Globus Project; www.globus.org
5. Goldberg, D. E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1999.
6. Axelrod, R., The evolution of strategies in the Iterated Prisoners Dilemma, in Genetic Algorithms and Simulated Annealing (L. Davies ed.), Morgan Kaufmann, pp. 32–41, 1987.
7. Pratihar, D., Deb, K., and Ghosh, A., A genetic-fuzzy approach for mobile robot navigation among moving obstacles, Int. J. Approximate Reasoning, **20(2)**, pp. 145–172, 1999.
8. Sims, K., Artificial Evolution for Computer Graphics, Computer Graphics (Proc. SIGGRAPH '91), **25(4)**, pp. 319–328, 1991.
9. Fleming, P. J., Purshouse, R. C., Chipperfield, A. J., Griffin, I. A., and Thompson, H. A., Control Systems Design with Multiple Objectives: An Evolutionary Computing Approach, Workshop in the 15th IFAC World Congress, Barcalona, 2002.
10. Cantú-Paz, E., and Goldberg, D. E., On the Scalability of Parallel Genetic Algorithms, Evolutionary Computation, **7(4)**, pp. 429–449, 1999.
11. Kleinroack, L., UCLA Press Release, July 3rd 1969.
12. Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S., The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Services Infrastructure WG, Global Grid Forum, June 22nd 2002.
13. Web Services Architecture, www.w3c.org/TR/ws-arch, W3C Working Group Note February 11th 2004.
14. Chappell, D. A., and Jewell, T., Java Web Services, O'Reilly, 2002.
15. Alander, J. T., Indexed Bibliography of Distributed Genetic Algorithms Technical Report 94-1-PARA, University of Vaasa, 2003.
16. Deb, K., Zope, P., and Jain, A., Distributed Computing of Pareto-Optimal Solutions with Evolutionary Algorithms, Proc. EMO 2003, pp. 534–549, Springer-Verlag, 2003.

17. Van Veldhuizen, D. A., Zydallis, J. B., and Lamont, G. B., Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms, IEEE Trans. on Evolutionary Computation, **7(2)**, pp. 144–173, 2003.
18. Tanimura, Y., Hiroyasu, T., Miki, M., and Aoi, K., The System for Evolutionary Computing on the Computational Grid, Proc. IASTED 14th Intl. Conf. on Parallel and Distributed Computing and Systems, pp. 39–44, ACTA Press, 2002.
19. Abdalhaq, B., Cortes, A., Margalef, T., and Luque, E., Evolutionary Optimization Techniques on Computational Grids, Proc. ICCS 2002, pp. 513–522, Springer-Verlag, 2002.
20. Luna, F., Nebro, A. J., and Alba, E., A Globus-Based Distributed Enumerative Search Algorithm for Multi-Objective Optimization Technical Report LCC 2004/02, University of Malaga, 2004.
21. Colan, M., Service Oriented Architecture expands the vision of Web Services: part 1, IBM developerWorks paper, 2004.

A Grid Architecture for Comfortable Robot Control

Stéphane Vialle[1], Amelia De Vivo[2], and Fabrice Sabatier[1]

[1] Supelec, 2 rue Edouard Belin, 57070 Metz, France
Stephane.Vialle@supelec.fr, sabatier_fab@metz.supelec.fr
[2] Universitá degli Studi della Basilicata,
C.da Macchia Romana, 85100 Potenza, Italy
devivo@unibas.it

Abstract. This paper describes a research project about robot control across a computing Grid, first step toward a Grid solution for generic process control. A computational Grid can significantly improve remote robot control. It can choose at any time the most suitable machine for each task, transparently run redundant computations for critical operations, adding fault tolerance, allowing robotic system sharing among remote partners.

We built a Grid spanning France and Italy and successfully controlled a navigating robot and a robotic arm. Our Grid is based on the GridRPC paradigm, the DIET environment and an IPSEC-based VPN. We turned some modules of robotic applications into Grid services. Finally we developed a high-level API, specializing the GridRPC paradigm for our purposes, and a semantics for quickly adding new Grid services.

1 Motivations and Project Overview

This paper introduces a Grid architecture for comfortable and fault tolerant robot control. It is part of a larger project aiming to develop a grid solution for generic process control.

Motivations. In several real situations robots must be remotely controlled. This is because we install robots where an application requires them. It can be a not computer-suitable environment, for example, for temperature constraints. Sometimes the building where the robots are is far away from that one where the computing centre is. In such a case probably the computer maintenance team is in the computing centre and it could be uncomfortable and expensive to have some computers near the robots.

Simple robots, like robotic arms, just need to receive commands and sometimes to send feedback. A simple remote application can manage the situation, but a devoted machine makes sense only if the robot is continuously used. An integrated environment, like a Grid environment, can run the robotic application on the first available computer when needed.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 344–353, 2005.

Complex robots, like navigating robots, are equipped with sensors and devices acquiring data about the surrounding environment. They send these data to a remote server running complex computations for deciding robot behavior. Navigating robot applications sometimes, but not always, need a powerful machine. A computational Grid could find a suitable machine on the fly, avoiding to devote an expensive computer to the robotic system.

Finally, a computational Grid can be useful and comfortable in several other situations. It can automatically switch to an unloaded computer when the current one gets overloaded, guaranteeing some QOS to time-constrained robotic applications. Some applications are embarrassingly parallel and a Grid can offer a different server for each task. A single application execution is not fault tolerant for critical missions. A Grid can automatically run the same application on different machines, so that if one fails, another one can keep robot control. In a robotic research environment, a Grid allows remote partners to easy share a robotic system.

Project Roadmap. In 2002 we started a four-step project about remote robot control across a Grid. A careful evaluation about delay, security policies, Internet uncertainty and so on [4, 6] was mandatory and the first step was about it. We worked with a *self-localization* application for a single navigating robot [7], using "ssh links" and a simple client-server mechanism. We focussed on ad-hoc overlap techniques for amortizing the Internet communication, and at the end we achieved a reasonable slow down [8]. In the second step we built and experimented a Grid based on a secure VPN and DIET [1]. We added a *navigation* module, we turned both modules into Grid services, and we designed and implemented an easy-to-use and easy-to-expand robotic API [2]. In our Grid configuration the robot, a client and some computing servers were in a single site in France, while another server was in Italy. The third step is a working in progress. Using our API we quickly and easily added a *lightness detection* Grid service for environment checking. Then we extended the Grid with other two sites in France, each one hosting just a computing server. Finally we added a second robot (a robotic arm) and a related control module. In the fourth step we will investigate the way to adapt our software architecture to the Globus middleware. The current solution is very suitable for our applications, but Globus is an example of more generic and standard middleware.

2 Robotic Applications and Grid Testbed

Hardware Resources. The physical resources on our Grid are two robots, some PCs at Metz Supelec campus, two PCs in two different sites in Metz and a PC at Salerno University, see figure 1. Our robots are an autonomous navigating *Koala*, with several onboard devices, and a robotic arm. Both are connected to external servers through serial links. Each server is a devoted PC controlling basic robot behaviors. All robotic applications are clients of these servers.

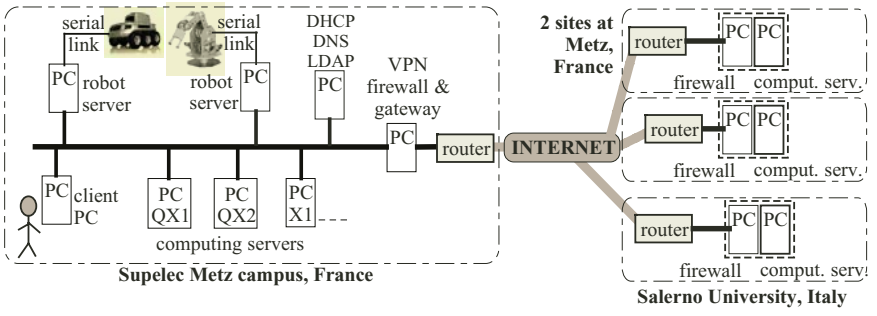


Fig. 1. Grid Testbed for robot control: 2 robots and four sites

Grid Environment. We chose the DIET [1] (Distributed Interactive Engineering Toolbox) Grid environment. It supports synchronous and asynchronous Grid-RPC calls [5], and can be considered a Grid Problem Solving Environment, based on a Client/Agent/Server scheme. A Client is an application that submits a problem to the Grid through an Agent hierarchy. This avoids the single Agent bottleneck. Agents have a Grid Servers list and choose the most suitable for a Client request, according to some performance forecasting software.

DIET communication is Corba-based, but this was a problem for our institutes security policies. In order not to relax our respective security levels, we created an IPSEC-based VPN [3]. This just requires the 500/UDP port opened and the ESP and AH protocols authorized on the destination gateway.

Robotic Testbed Applications. The testbed application for the *robotic arm* is a simple action loop, while the testbed application for the *Koala* robot consists of three complex modules: self-localization, navigation and lightness detection. Our robot navigates in dynamic environments, where no complete predetermined map can be used. Artificial landmarks are installed at known coordinates. When switched on, the robot makes a panoramic scan with its camera, detects landmarks and self-localizes [7]. Based on its position, it can compute a theoretical trajectory to go somewhere. For error compensation, new self-localizations happen at intermediate positions. During navigation the robot checks the environment lightning and, eventually, signals problems. For this purpose it moves its camera and catches images. The *Koala Server* always sends its clients JPEG-compressed images.

3 Software Architecture and Grid Deployment

Figure 2 shows our Grid architecture. At the toplevel, a *Grid application* is almost like a classical application. It calls *RobGrid API* functions (see section 6) to achieve Grid services. Our Grid services implement robotic application modules, so that they appear as high-level robot commands, such as "Localization"

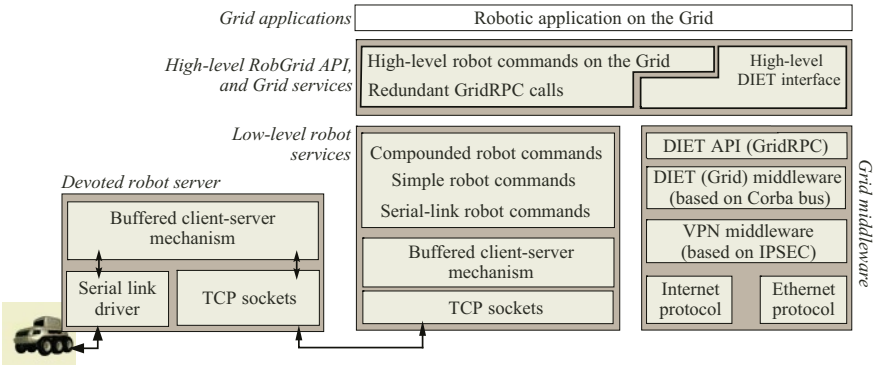


Fig. 2. Software architecture overview

or "Navigation". Users can call them concurrently, but in this case they are responsible for robot devices coordination and synchronization.

We implemented our Grid services using the DIET GridRPC interface, according to the *RobGrid API* semantic rules (see section 5). Depending on the service, it can make a synchronous call to a low-level service (to control just one robot device), an asynchronous call to a low-level service (to simultaneously control more robot devices), or several concurrent asynchronous calls to low-level services, for example to run redundant computations on different Grid servers. When a Grid service runs redundant computations, it waits just for the first one to finish, ignoring or cancelling the others. All these details are hidden to the programmer that can focus on robotic problems.

Each low-level robot service is implemented as a three layers stack. A *buffered client-server mechanism* allows different and/or redundant tasks to concurrently access a robot server through *TCP sockets*. This supports the fault tolerance strategy of our grid (see section 4). Finally a *serial link driver* runs on each devoted robot server.

The Grid middleware consists of the DIET API, a Corba bus and a secure IPSEC-based VPN. We configured IPSEC so that our grid has a main node at Supelec, including a LAN segment and a gateway-and-firewall PC. Each of the other three sites have just a stand alone computer, with a lighter configuration. It does not include a gateway, so that the stand alone PCs can only communicate with the main node. The client machine can access all Grid services only if it is on the main node. Of course this solution is not suitable for a larger Grid and we are currently working for removing this limitation.

IPSEC requires just the 500/UDP port for security keys exchanging, and the AH and ESP protocols for secure communication. This made possible to deploy an IPSEC-based VPN without changing our local security policies.

In figure 3 there is an example of (*Koala*) robot control across our Grid. The client application, on a client PC, requires some Grid services through the *RobGrid API*. The underlying DIET functions contact the DIET agents running

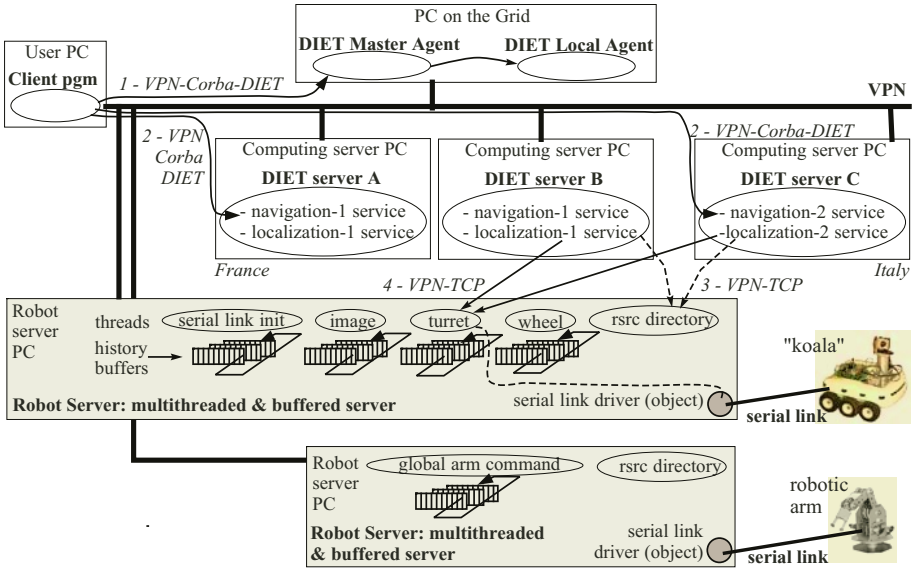


Fig. 3. An operation sequence example

somewhere on the Grid to know the addresses of the most suitable computing servers. Then the user program contacts them directly through the DIET protocol on the Corba bus, and each computing server establishes a direct communication with the *Koala Server*, using TCP sockets instead of the DIET protocol. This way camera images to be sent to the Grid servers can avoid Corba encoding. After processing, only small results (such as a computed localizations) has to be sent to the client machine across the Corba bus.

4 Low-Level Robot Services

Devoted Robot Servers. A robot is a set of independent devices (wheel, camera, infra red sensors, articulation, grip...), that can be considered as a set of independent resources accessible through related elementary or low-level services.

A devoted robot server collects all elementary services for its connected robot. Each service is attached to a different port (see the bottom of figure 3) and is multithreaded, so that it can serve several clients. If necessary, it can also lock the related resource. For example, several clients (actually Grid servers) can simultaneously connect to the camera for taking images while the robot navigates, but only one client at a time can drive the camera motor.

Resource Directory Service. In order to make the Grid servers independent from the robot server, we added a *resource directory service* (see the bottom of

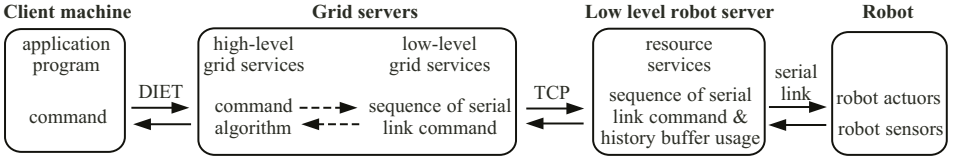


Fig. 4. Translation steps from an application command to robot device commands

figure 3). Clients have just to know its port number for achieving the directory of all available resources and related ports. This way low-level robot services upgrades have no impact on Grid services.

History Buffers. Redundant computations require the same robot data for all redundant clients, so we associate a number of *history buffers* to each robot device. On figure 3 each robot resource has three buffers. Each Grid service has a default number of history buffers, but applications can modify it.

Before to reset a history buffer, we have to consider that several redundant Grid services can be using it. Each Grid service needs a *reset strategy* depending on the related robot device and on the Grid service algorithm. All reset strategies are implemented in the *RobGrid* API (see section 6).

Slow Grid servers asking for too old execution are rejected and finally cancelled. However, slow servers get results faster because they have not to wait for robot actions. So, few servers are actually rejected and cancelled. If the network load changes during the execution, the Grid server that sends the new next command may change too. In this case the robot server goes ahead, driven by the new fastest Grid server.

5 Grid Services

Grid services implement robotic application modules and, of course, we can add other of them in future. Our *RobGrid* API introduces a semantics for Grid service programming. This makes Grid service development easy for robotic researchers.

For adding a new Grid service it is not required to deal with low-level details and to manage the communication with the robot server. Grid services have just to implement four sub-services and to define a *reset strategy* for history buffers:

- **Connection.** This is done by a generic Grid service that contacts the resource directory service, asks for the required TCP port and connects the application to a low-level service.
- **Disconnection.** This is done by a generic Grid service that closes the application connection to a TCP port.

- **History Buffer Reset.** This is a simple reset request for one of the history buffers associated to the related device. An index identifies the required buffer, according to the buffer reset strategy of the Grid service.
- **Robotic Command Execution.** This is something like *navigation(x,y, theta)*. Grid services execute each high-level robot command calling the staked up low-level robot commands. Each stack layer decomposes a higher-level command in more simple commands. For example the navigation command above can include *move_straightforward(x,y)*. At the bottom of the stack there are basic commands (serial link commands) to be sent to the devoted robot server across TCP sockets. Here the serial link driver gets the robot to execute the lower-level commands. Figure 4 illustrates the translation steps from an application command to robot device commands.

We experimented our semantic rules adding a new Grid service. It implements the *lightness measurement* module for the navigating robot application. Then we extended our testbed with a robotic arm and a related application module. The same low-level robotic library drives both the robotic arm and the navigating robot. In order to integrate the arm application module we added a new Grid service and related low-level services. In both cases we encountered no major difficulties.

6 The RobGrid API

To make the robotic application development easy, our *RobGrid* API offers classes and objects for Grid service interfacing. They hide DIET communication details and automatically initialize some DIET data structures. The application has to explicitly call three sub-services of each Grid service: *connection*, *disconnection* and *command execution*. *RobGrid* provides very friendly functions for this purpose. The *buffer reset* sub-service, instead, is called in a transparent way, according to the predetermined *reset strategy* of the Grid service which is implemented in a *RobGrid* object.

When creating a local interface object, a user just specifies a redundancy factor. The local interface object chooses the requested Grid servers, waits for the first one to finish, cancels the others, and returns the results.

Moreover, it is possible to request a Grid service through a local interface object in an asynchronous way. The API provides functions for testing and waiting for a service completion. See [2] for details about *RobGrid* API usage.

7 Experimental Results

Experimentation on a Local Sub-Grid. In real conditions, researchers working on new control algorithms are near the robotic system, for avoiding unexpected problems. A comfortable solution is to use a laptop, a wifi connection and

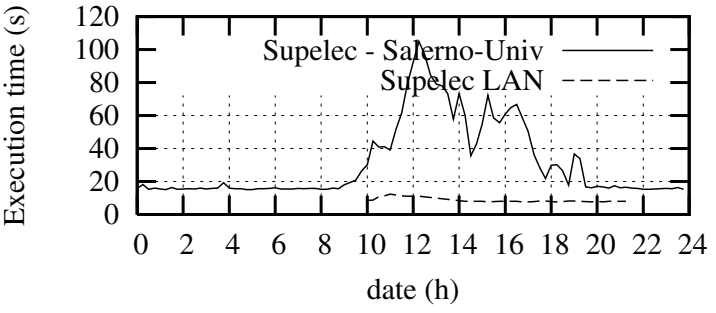


Fig. 5. Limited slow down across the Internet

a computing server for large computations. The wifi connection allows to control the robot and to walk around the robotic system when needed, and a computing server avoids to overload the laptop. But generally a computing server is a shared resource and can be loaded. A local grid including several servers can choose an unloaded machine on the fly. It can also run redundant computations on different servers and get the first available result.

Table 1 shows experimental performance on a local sub-grid for the self-localization module. We run redundant computations for critical tasks on several servers and wait for the first to finish. Execution time decreased until 9.5-8.5s against 13.5s measured with a single shared server. Last columns of table 1 show there is no remarkable overhead using DIET on this benchmark.

Performance Across the Internet. We measured the self-localization performance when the localization service runs in Italy. During 24 hours its average execution time elapsed from 15.5s during the night up to 100s during the day (see figure 5). In both cases remote localization succeeded and during the night the slow down was limited to a 2 factor compared to a local computation. So, running robot control services in Italy can be an interesting solution during the night.

Fault Tolerance. We experimented a complete long application for the navigating robot, with several localization and navigation steps, on the whole Grid. Robot camera was simultaneously controlled by two Grid servers, one in France

Table 1. Execution time on a local sub-grid for the self-localization service. *Devoted resources* and *overloaded laptop* are not real situations

Laptop-wifi		Laptop-wifi+local sub-Grid		Laptop-wifi+devoted rsrcs	
Basic pgm (Std load on the laptop)	Optimized pgm (Overloading the laptop)	One Grid server	N redundant Grid servers	One server across an unloaded Grid	One server across a ssh link
14.34s	11.23s	**13.5-10.5s**	**9.5-8.5s**	8.65s	8.65s

Table 2. Execution time of the localization Grid service on different nodes

Grid machines	Computing PC server at Supelec (France)	Desktop PC at Metz (France)	Computing PC server at Salerno University (Italy)
day run	9.5s-8.5s	**21.5s-23.7s**	$\approx$ 100s (large variations)
night run	9.5s-8.5s	21.5s-23.7s	$\approx$ **15.5s**

and the other in Italy. We killed the local server and the robot camera continued to execute its panoramic scans, slower, controlled by the remote server from Italy. Then we run again the server in France, and the client application speeded-up. So we achieved a fault tolerant behavior when a server went down, avoiding the robot mission failure.

Even if the localization slow down was significant during the day, it was limited to some parts of the application and had a reasonable global impact. The whole application slow down was limited to a 2.5 factor during the day ($252s$ instead of $102s$).

First Scalability Experiment. To test the scalability of our Grid architecture, we added a new module to the navigating robot application (lightness measurement service), a new robot (a robotic arm) and 2 new nodes in the grid (2 PCs in two different areas in Metz). From a configuration point of view, we had a little trouble to extend the VPN and the Corba bus to PCs with dynamic IP addresses. Grid gateway and firewall configuration has to be updated each time an IP address changes.

About performance, we registered no slow down when our servers controlled both the robots simultaneously. Robot localization on servers from the new nodes took between 21.5s and 23.7s during the day. These nodes have low-speed connections, that is download at 512 KB/s and upload at 128 KB/s. They appeared interesting solution for redundant computations during the day, while the Italian server is a better solution during the night (see table 2). A larger Grid with several nodes improves the capability to find computing resources at any time with limited slowdown.

8 Conclusion and Perspectives

During the first steps of this project we built and experimented a secure Grid for robot control. We developed a friendly API for application programmers, a Grid semantics for high-level service developers, and low-level services supporting concurrent and redundant requests to robot devices. We obtained a comfortable and efficient environment for robotic experiments, and, finally, we showed that this Grid is easy to extend.

Currently we are working for adding new robots on different sites, and for allowing client applications to run from any Grid node. Next step is a porting of our software architecture on a more standard grid middleware, like Globus.

But our final goal remains a more generic Grid for *process control*, allowing researchers and engineers to easily share physical processes and related computing resources.

Acknowledgements

This research is partially supported by Region Lorraine and ACI-GRID ARGE research project.

Authors want to thank Hervé Frezza-Buet for low-level robot library development, Alexandru Iosup for optimized versions of the navigating robot application, and Yannick Boyé for preliminary implementation on the DIET environment.

References

1. F. Lombard J-M. Nicod M. Quinson E. Caron, F. Desprez and F. Suter. A scalable approach to network enabled servers. *8th International EuroPar Conference, volume 2400 of Lecture Notes in Computer Science*, August 2002.
2. A. De Vivo F. Sabatier and S. Vialle. Grid programming for distributed remote robot control. *13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004). Workshop on Emerging Technologies for Next Generation GRID*, June 2004. Modena, Italy.
3. I. Foster and C. Kesselman. *N. Doraswamy and D. Harkins. Ipsec: The New Security Standard for the Inter- net, Intranets, and Virtual Private Networks.* Prentice-Hall, 1999.
4. L. Frangu and C. Chiculita. A web based remote control laboratory. *6th World Multiconference on Systemics, Cybernetics and Informatics*, July 2002. Orlando, Florida.
5. S. Matsuoka J. Dongarra C. Lee K. Seymour, H. Nakada and H. Casanova. Overview of gridrpc: A remote procedure call API for grid computing. *Grid Computing - GRID 2002, Third International Workshop Baltimore, Vol. 2536 of LNCS*, November 2002. Manish Parashar, editor, MD, USA.
6. S.H. Shen R.C. Luo, K.L. Su and K.H. Tsai. Networked intelligent robots through the internet: Issues and opportunities. *Proceedings of IEEE Special Issue on Networked Intelligent Robots Through the Internet*, 91(3), March 2003.
7. A. Siadat and S. Vialle. Robot localization, using p-similar landmarks, optimized triangulation and parallel programming. *2nd IEEE International Symposium on Signal Processing and Information Technology*, December 2002. Marrakesh, Morocco.
8. A. De Vivo and S. Vialle. Robot control from remote computers through different communication networks. *Internal Report*, January 2003.

The Grid-Ireland Deployment Architecture

Brian Coghlan, John Walsh, and David O'Callaghan

Department of Computer Science,
Trinity College Dublin, Ireland
coghlan@cs.tcd.ie, john.walsh@cs.tcd.ie
david.ocallaghan@cs.tcd.ie

Abstract. Grid-Ireland is unusual in its integrated infrastructure, and the stress that is laid on homogeneity of its core. The major benefit is the decoupling of site details from the core infrastructure, and the resulting freedom for heterogeneity of site resources. We describe the efforts to support this heterogeneity in a systematic way. We also describe the deployment architecture and a methodology to increase the availability of the core infrastructure.

1 Introduction

The HPC facilities in Ireland are very limited. There is an Origin 3800/40, a 64-way Xeon cluster and a 6TB disk farm at NUI, Galway, and a 20-CPU SGI Altix 3700 has just been installed. There is a 100-CPU cluster in the Boole Centre at UCC in Cork, and two 96-CPU clusters plus funding for three extra 96-CPU clusters in the NMRC at UCC. There are 80-CPU P3 and 130-CPU Xeon clusters and a 4TB disk farm at TCD in Dublin, and a fully immersive VR cave is to be installed. A 256-CPU cluster will be installed at UCD in Dublin by the end of 2004. There is a 84-CPU cluster at NUIM in Maynooth. There are several small-scale clusters. There is funding from Science Foundation Ireland (SFI) and the Higher Education Authority (HEA) for several medium-scale clusters (100-500 CPUs) and two medium-scale data farms in 2004/5.

Whilst the experimental and theoretical science paradigms remain strongly embedded in Irish science, there is strong growth in the hybrid paradigm, computational science. Most of this scientific computing is still done on local facilities. It involves a wide range of application areas, but few truly parallel applications. Most users develop their codes but use commercial libraries and tools. The reference architectures for these are a major factor in the choice of HPC architecture, i.e. most of the deployed architectures are mission-specific.

Currently there is no large-scale facility in Ireland. Until very recently there was no identified governmental intention to have one. In August 2004, however, SFI announced that they wished to enhance the high-end computational capabilities of the overall Irish research community by the creation of a National Centre for High End Computing within the Republic of Ireland. Phase 1 funding has been approved and Phase 2 funding will follow in 2005. It is very likely that the resulting centre will include a mix of mission-specific architectures.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 354–363, 2005.

These limited and mostly mission-specific resources should not be wasted by being inaccessible to the Grid simply because their architectures are not those of the reference ports.

1.1 Grid-Ireland

Grid-Ireland provides grid services above the Irish research network, allowing researchers to share Irish computing and storage resources using a common interface. It also provides for international collaborations by linking Irish sites into the European grid infrastructures being developed under such EU projects as EGEE, LCG and CrossGrid. The grid infrastructure is currently based on LCG2, the common foundation that ensures interoperability between participating scientific computing centres around the world. Internationally, members of Grid-Ireland are involved in the EU EGEE, CrossGrid, JetSet and COST 283 iAstro projects, and there are links to the UK GridPP and e-Science programs. The Grid-Ireland OpsCentre is the EGEE Regional Operations Centre (ROC) for Ireland.

Grid-Ireland currently encompasses six sites, at TCD, UCC, NUIG, DIAS, UCD and QUB, with an Operations Centre in TCD. It aims to make Grid services accessible to an additional eleven Irish third-level institutions in the near future as a result of a generous donation by Dell Ireland Limited. The Irish NREN (HEAnet) are substantively assisting in this initiative. Grid-Ireland will then encompass 17 sites, i.e. the majority of academic institutions in Ireland will be connected to the national grid.

There are three major national VOs. The first, CosmoGrid, is a collaborative project entitled Grid-enabled computational physics of natural phenomena, explicitly aimed at inter-institutional and interdisciplinary compute-intensive research in natural physics, with nine participating institutions, led by the Dublin Institute for Advanced Studies (DIAS). Astrophysics are a key and central element of the project centred on astrophysical objects ranging from supernova remnant (with strong collisionless shocks), forming stars (jets and outflows) to neutron stars (radiative processes) and the sun (the solar transition region). In addition to those areas, studies on gravitational waves, adaptive optics, meteorology (regional climate models), geophysics (full simulation of a digital rock) and atmospheric physics are being pursued.

The second VO, MarineGrid, is a data-intensive collaboration between Geological Survey of Ireland, Marine Institute and four Universities (NUIG, UCC, UCD and UL). The Irish National Sea-bed Survey is taking place at present through bathymetric mapping of the seabed. Ireland is an island with nine tenths of its area under water. The seabed survey spans 525,000km^2 and currently contains approximately 6TB of data. Detailed knowledge of the seabed topography with location resolutions of up to 2m will have significant economic implications on, for example, fishing and mineral resource management. An unexpected spin-off is exploitation of historically valuable wrecks. The Survey is a valuable government resource with associated security concerns.

The third VO, WebCom-G, is investigating an alternative to existing von Neumann grid execution models, which are not appropriate to their high-latency, loosely-coupled infrastructure. UCC, TCD, NUIG and QUB are creating a condensed graph grid engine that exploits laziness and speculation and is compatible with and uses traditional grids. There is a depth of interest in Irish computer science circles about issues of languages, programming models[1] and execution models[2][3] for heterogenous environments, and this VO is a good example. Grid-Ireland specifically wishes to support these research directions.

1.2 Homogeneous Core Infrastructure, Heterogenous Resources

There are three further motivations:

(a) To minimize the demand on human resources by minimizing the proportion of the software that needs to be ported. The simplest component of most grid software frameworks is that relating to the worker nodes.
(b) To minimize the demand on human resources by maximizing the proportion of the software that does *not* need to be ported. Thus all the non-worker node components should use the reference port, i.e. the core infrastructure should be homogeneous.
(c) To maximize the availablility of the infrastructure. Grid-Ireland has designed a transactional deployment system to achieve this[4]. This requires that the core infrastructure be homogeneous, and also centrally managed.

Thus *Homogeneous Core Infrastructure, Heterogenous Resources* is a pervasive motto that encapsulates explicit and implicit principles:

(a) *Explicit homogeneous core infrastructure:* this principle enables a uniform dedicated core national grid infrastructure, which supports a uniform architecture based on reference ports of the grid software, and thereby frees resources for maximum focus on the critical activites such as security and monitoring/information systems. Logically, it allows a uniform control of the grid infrastructure that guarantees uniform responses to management actions. It also assures a degree of deterministic grid management. Furthermore it substantially reduces the complexity of the release packaging and process. Minimizing this complexity implies maximizing the uniformity of the deployed release, i.e. the core infrastructure should be homogeneous.
(b) *Implicit centralized control via remote management:* this principle enables simpler operations management of the infrastructure. Remote systems management enables low-level sub-actions to be remotely invoked if required. It also enables remote recovery actions, e.g. reboot, to be applied in the case of deadlocks, livelocks, or hung hardware or software. Realistically, all infrastructure hardware should be remotely manageable to the BIOS level.

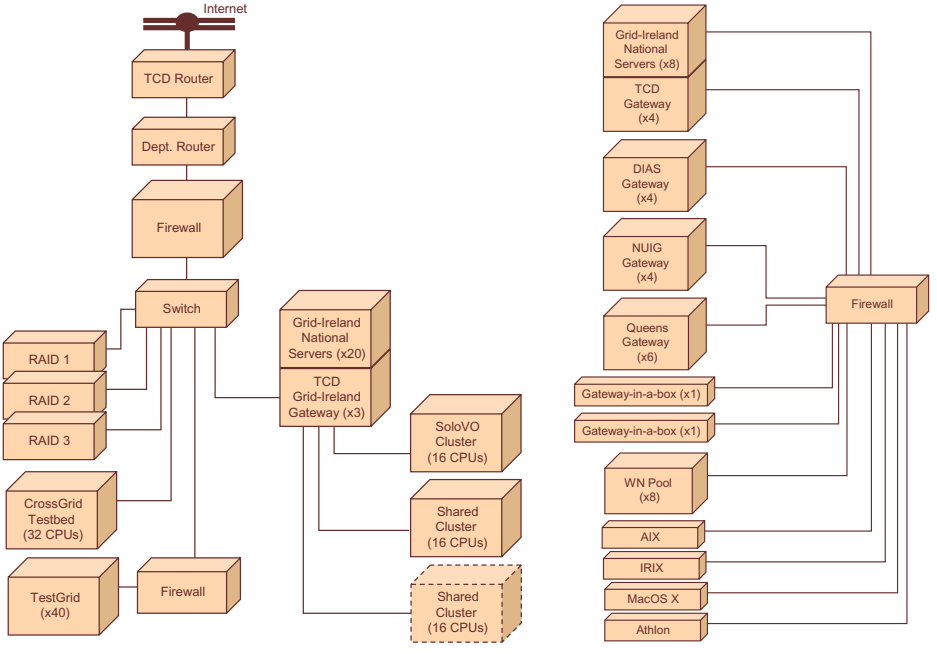


Fig. 1. (a) OpsCentre resources (b) TestGrid

(c) *Implicit decoupling of grid infrastructure and site management:* this principle enables the infrastructure and sites to be independent. It can encompass policies, planning, design, deployment, management and administration. In particular it allows the infrastructure upgrade management to be independent of that of the site, and non-reference mission-specific architectures to be deployed at the site.

Grid-Ireland has been designed with this approach since mid-2001. Funding was sought and eventually granted, a senior Grid Manager appointed, a Grid Operations Centre established and staffed, infrastructure specified, purchased and installed, and finally middleware and management tools deployed. We consider that the use of these principles has been highly beneficial.

The Operations Centre resources are illustrated in Figure 1(a). They include approximately 20 national servers, 64-CPUs and a 4TB disk farm for the various testbeds, and a certification TestGrid. The TestGrid (see Figure 1(b)), which includes approximately 40 machines and a 4TB disk farm, serves multiple purposes: it implements a fully working replica of the national servers and sites; it permits experimentation without affecting the national services; it acts as the testing and validation platform; and it acts as a non-reference porting platform.

2 Heterogeneity

Grid-Ireland wished, in the first instance, that the porting of the LCG2 software to other platforms would focus on the ability to execute Globus and EDG jobs on worker nodes, and that replica management, R-GMA and VOMS would be supported. There was also a desire that MPI, replica management and the OpenPBS client be provided on each worker node. In some cases Torque might be required since newer versions of operating systems are not always provided for in OpenPBS. Also the R-GMA information system producer and consumer APIs and the VOMS client were required.

In summary we wished to port:

1. VDT
2. MPI
3. OpenPBS or Torque client
4. R-GMA producer and consumer APIs
5. VOMS client

There are a number of on-going issues, but we have successfully ported the functionality for job submission to Fedora Core 2, IRIX 6.5.14 and 6.5.17m, AIX 5.2L and Red Hat 9. We also plan to do this for Mac OS X v10.3 very soon, and a number of other platforms if the need arises within Grid-Ireland.

A number of CVS repositories are used to build all the necessary software for a worker node. The head version of VOMS is obtained from INFN's own repository. The whole of LCG2 is extracted using CVS checkouts directly from CERN's lcgware repository. The CrossGrid software is obtained by directly copying the CVS repository to a local repository. Nightly builds are then done from this local repository. The RAL repository of R-GMA will also need to be added soon, since LCG2 no longer maintain the most recent version of R-GMA.

Figure 2 shows the status of the build system in November 2004. The results change quite regularly as new ports are completed.

OS Type	Version	VDT	Basic	VOMS	RGMA	RM
Redhat	7.3	RPMS	RPMS	RPMS	RPMS	RPMS
Redhat	9.0	RPMS	RPMS	RPMS	RPMS	RPMS
Fedora Core	2	RPMS	RPMS	RPMS	tarball	tarball
					RPMS	RPMS
SGI	6.5.14	tarball	tarball	tarball	tarball	tarball
AIX	5.2L	tarball	tarball	tarball	tarball	tarball
Darwin	10	tarball	tarball	tarball	tarball	tarball

Colour	Meaning
	To be started
	Started
	Done

Fig. 2. Auto-build Results for Worker Nodes

3 Deployment

As stated above, Grid-Ireland has installed a grid computing infrastructure that is fully homogeneous at its core. Each of the sites connects via a grid gateway. This infrastructure is centrally managed from Trinity College Dublin. These gateways are composed of a set of seven machines: a firewall, a LCFGng install server, a compute element (CE), a storage element (SE), a user interface machine (UI), a worker node that is used for gateway tests only, and optionally a network monitor. All the sites are identically configured. The grid software is initially based on LCG2, but later will follow the EGEE releases.

As can be seen from Figure 3, the site resources (shown as a cluster of worker nodes) are outside the domain of the gateway; these resources belong to the site, and the site is always in charge of their own resources. One of the key departures from the structures for deployment commonly used in Europe is to facilitate those resources to be heterogeneous with respect to the gateways. As explained in Section 2, Grid-Ireland is attempting to provide ported code for any potential platform that will be used for worker nodes (WNs). The rationale behind this is described in the early sections of this paper.

The only requirement on the site is that the worker nodes be set up to cater for data- and computation-intensive tasks by installing the worker-node software outlined in Section 2 This includes both the Replica Management software from LCG2 and the various versions of MPICH from CrossGrid. A site submits jobs via the gateway UI, whilst the gateway CE exports core grid services to the site and queues job submissions to the site resource management system, and its SE provides scratch storage for grid jobs. Therefore the gateway is both the client of the site and vice versa.

Grid-Ireland has specified its gateways to ensure minimal divergence from standard site configuration, minimal hardware and space costs per site, and minimal personnel costs per site. The basic technology for this is the use of virtual machines. Currently there are two physical realisations of this architecture. At minimum, a generic Grid-Ireland gateway comprises a single physical machine, a switch, and a UPS unit. The machine runs its own OS plus a number of virtual machines that appear to be a normal machine both to the host OS and to external users. The Linux OS and grid services are remotely installed, upgraded and managed by the Operations Centre, without needing any attention at the site. The firewall and LCFG server run concurrently on the host operating system. All other servers are hosted as virtual machines. Eleven such gateways are presently being prepared for deployment.

For more demanding sites the functionality is spread over four physical machines, with the firewall, LCFG server, CE and SE running on the host operating systems. The other servers are hosted as virtual machines: the test WN on the CE, and the UI and NM on the SE. Six such gateways are already deployed.

Apart from the firewall, all other servers on the gateways are installed via PXE from the LCFG server. The LCFG server itself is manually installed, but thereafter it is updated with new releases from the central Grid-Ireland CVS repository, see Figure 4. It is usual that this is a manual process involving CVS

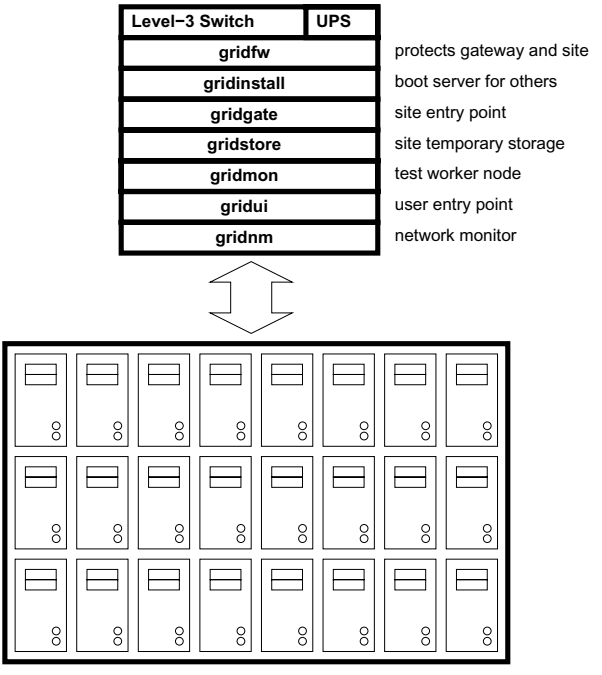


Fig. 3. Generic Grid-Ireland Site

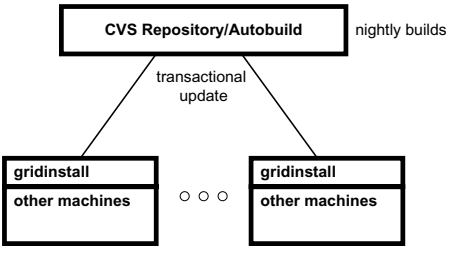


Fig. 4. Deployment Process

checkouts. Whilst this takes place the site is essentially in an inconsistent state. Grid-Ireland, however, have designed an automatic process that is specifically intended to reduce inconsistency to a minimum.

3.1 Consistency

Let us consider for instance that a new release of some infrastructural grid software is incompatible with the previous release, as is often the case. Once a certain proportion of the sites in a grid infrastructure are no longer consistent with the new release then the infrastructure as a whole can be considered inconsistent. Each grid insfrastructure will have its own measures, but in all cases

there is a threshold below which proper operation is no longer considered to exist. The infrastructure is no longer available. Thus availability is directly related to consistency. An inconsistent infrastructure is unavailable.

The average time a site waits before it becomes consistent is called the mean time to consistency MTTC. The interval between releases MTBR is quite independent of the MTTC. The MTTC is a deployment delay determined by the behaviour of the deployers, whilst the MTBR is dependent upon the behaviour of developers.

3.2 The Need for Transactionality

Maximizing availability means maximizing the proportion of time that the infrastructure is entirely consistent. This requires either the the time between releases MTBR to be maximized or the MTTC to be minimized. The MTBR is beyond the control of those who manage the infrastructure. On the other hand, if the MTTC can be minimized to a single, short action across the entire infrastructure then the availability will indeed be maximized.

However, an upgrade to a new release may or may not be a short operation. To enable the upgrade to become a short event the upgrade process must be split into a variable-duration prepare phase and a short-duration upgrade phase, that is, a two-phase commit. Later in this paper we will describe how this can be achieved.

If the entire infrastructure is to be consistent after the upgrade, then the two-phase commit must succeed at all sites. Even if it fails at just one site, the upgrade must be aborted. Of course this may be done in a variety of ways, but from the infrastructure managers' viewpoint the most ideal scenario would be that if the upgrade is aborted the infrastructure should be in the same state as it was before the upgrade was attempted, that is, the upgrade process should appear to be an atomic action that either succeeds or fails.

Very few upgrades will comprise single actions. Most will be composed from multiple subactions. For such an upgrade to appear as an atomic action requires that it exhibits transactional behaviour, that all subactions succeed or all fail, so that the infrastructure is never left in an undefined state.

Thus we can see that to maximize availability requires that an upgrade be implemented as a two-phase transaction.

3.3 Transactional Deployment System

We have implemented transactional deployment using three components[4]. The first is a repository server, which hosts both the software to be deployed and the Transactional Deployment Service (TDS) logic. Secondly, there is a user interface, which has been implemented as a PHP page residing on an Apache web server. Finally there are the install servers at the sites that we are deploying to. These servers hold configuration data and a local copy of software (RPMs) that are used by LCFGng to maintain the configuration of the client nodes at the site. It is the state of these managed nodes that we are trying to keep consistent.

Table 1. Example MTBR, MTTC and availability for transactional deployment

Estimated MTBR	163 hours
Estimated MTTC	17.5 minutes
Estimated availability	99.82%

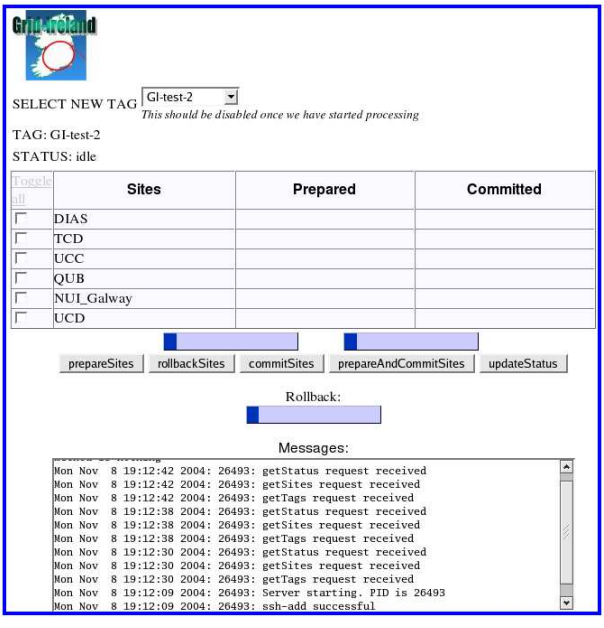


Fig. 5. Transactional Deployment System GUI

It will be a while before statistically significant data is available for transactional deployment to a real grid infrastructure such as Grid-Ireland. The MTBR, i.e. the time between releases, is the same with or without transactional deployment. We have estimated the worst-case MTTC as 17.5 minutes, and if we assume the CrossGrid production testbed MTBR by way of example, then the resulting worst-case infrastructure availability is as shown in Table 1.

4 Conclusions

Grid-Ireland is unusual in its integrated infrastructure, and the stress that is laid on homogeneity of its core. The principles behind this have been described above. The major benefit is the decoupling of site details from the core infrastructure, and the resulting freedom for heterogeneity of site resources.

We have described our efforts to support this heterogeneity by porting to non-reference platforms in a systematic way, with nightly autobuilding. This has allowed us to begin a most interesting set of benchmarking and heterogeneity experiments involving all of these platforms.

It is clear that the infrastructure has greatly enhanced availability with transactional deployment. However, the most important benefit of the transactional deployment system is the ease it brings to deployment. Transactional deployment allows for an automated totally-repeatable push-button upgrade process (see Figure 5), with no possibility of operator error. This is a major bonus when employing inexperienced staff to maintain the grid infrastructure.

Acknowledgements

We would like to thank Enterprise Ireland, the Higher Education Authority, Science Foundation Ireland and the EU for funding this effort. We gratefully thank DIAS for the SGI machine they have loaned to us for the IRIX port, and IBM and Dell for sponsoring us with machines to perform ports to their platforms. Most of all we would like to thank those at INFN and CERN for all their help in porting to each platform.

References

1. Lastovetsky, A.: Adaptive parallel computing on heterogeneous networks with mpc. Parallel Comput. **28** (2002) 1369–1407
2. Ryan, J.P., Coghlan, B.A.: Smg: Shared memory for grids. In: The 16 th IASTED International Conference on Parallel and Distributed Computing and Systems, Cambridge, MA, USA (2004)
3. Morrison, J.P.: Condensed Graphs: Unifying Availability-Driven, Coercion-Driven, and Control-Driven Computing. PhD thesis, Technische Universiteit Eindhoven (1996)
4. Coghlan, B., , Walsh, J., Quigley, G., O'Callaghan, D., Childs, S., Kenny, E.: Principles of transactional grid deployment. European Grid Conference (2005)

UNICORE as Uniform Grid Environment for Life Sciences

Krzysztof Benedyczak[1], Michał Wroński[1], Aleksander Nowiński[2], Krzysztof S. Nowiński[2], Jarosław Wypychowski[2], and Piotr Bała[1,2]

[1] Interdisciplinary Center for Mathematical, and Computational Modelling, Warsaw University, Pawinskiego 5a, 02-106 Warsaw, Poland
[2] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Chopina 12/18, 87-100 Toruń, Poland

Abstract. In this paper we present UNICORE middleware as uniform framework for life sciences applications. The UNICORE provides environment for integration of the different tasks as simulations, database access or data acquition. We have developed number of application specific interfaces which can substitute existing visualization packages. The significant advantage of this solution is its modularity which makes development of new interfaces fast and easy. The UNICORE middleware allows user use these components in the grid environment by providing access to the distributed geographically resources.

1 Introduction

Life sciences become one of the most rapidly growing research fields. Molecular biology and quantum chemistry traditionally provide large number of nontrivial, important and computationally intensive problems. Recently, development of the experimental techniques provided research community with the large number of sequence or micro-array data. Because of the complexity of the studied systems the significant computational effort must be taken to provide scientific insight. Lack of the realistic model of the studied systems makes this tasks very difficult. Another characteristic feature of the life sciences research is large number of applications, tools and standards involved. The user has to access different databases, transform data and use number of applications which requires complicated workflow and dataflow management.

The users are forced to search for computational systems who can carry out simulations and, which is sometimes even more important, are available to the users. They have to move large amount of data from local systems to the remote one and vice versa. Unfortunately, the resources such as computational servers, databases and data acquition systems are not available at the user workstation. The users sharing external as well as departmental or even local computers are

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 364–373, 2005.

faced with the different site polices and practices such as different security, different user identification, different priorities or scheduling mechanisms and so on.

The growing group of the researches in the life sciences research very often has limited background in the computational chemistry and biology and is computer experience is limited to the use of the desktops. The users, especially ones in the life sciences research, expect intuitive tools which are easy to learn and use. Such tools cannot be just another application, but rather should allow for integration of existing tools and programs within one uniform framework.

Recent advances in the computer technology, especially grid tools make them good candidate for development of the uniform interface to distributed resources [1]. Computational grids enable sharing a wide variety of geographically distributed resources and allow selection and aggregation of them across multiple organizations for solving large scale computational and data intensive problems in chemistry and biology. This includes access to various databases as well as unique experimental facilities. In order to attract users, grid interfaces must be easy to install, simple to use and able to work with different operating system and hardware platforms.

2 Grid Middleware

Grid middleware is designed to provide access to remote high performance resources. However most of the development goes to the design and production of general tools which can be used in different situations. The existing solutions like Globus [2], Legion [3], LSF [4] address most important issues. Unfortunately, these tools require an advanced computer skills for the installation and usage. Moreover, in order to provide required functionality user application must be modified, sometimes significantly. This is not possible for commercial or legacy codes which cannot be easily adopted by the user, however, such programs are most widely used.

In order to overcome these disadvantages, the web interfaces are being developed. This approach is represented by the portals dedicated to the particular software package where user can prepare job, submit and control it. A good example is BioCore [5] which additionally provides user with quite advanced collaborative environment.

One should note, that advantages of the existing prototypes of the web based interfaces to the grid middleware resulted in the proposition of the Open Grid Software Architecture (OGSA) [6] which, amongst other advantages, should allow for much easier development of the users interfaces.

This approach is presented by the European DataGrid [7] which provides user with the web access to the selected applications in the area of bioinformatics or image processing. User obtains web interface to the distributed computational resources and can prepare and submit job. Results are transferred to the users workstation with the web browser and can be saved for further processing. The underlying grid middleware which provides access to the computational resources is based on the Globus Toolkit.

Different approach motivated development of the UNICORE [8] middleware. Compared to other tools UNICORE provides access top the various types of resources, has advanced user interface, is more flexible and allows for much easier integration with external applications. The details of the UNICORE can be found elsewhere [9] and we will summarize here its most important features and present recent achievements and extensions.

2.1 UNICORE Middleware

The UNICORE architecture is based, like other grid middleware, on the three tier model. It consists of the user, server and target system tier (see Fig. 1). The user tier consists of the graphical user interface, the UNICORE client, written as Java application. It offers the functions to prepare and control jobs and to set up and maintain the user's security environment. The UNICORE job can be build from multiple parts which can be executed asynchronously or dependently on different systems at different UNICORE sites. Within the UNICORE environment user has a comfortable way to use distributed computing resources without having to learn about a site or system specifics.

The UNICORE security is based on the Secure Socket Layer (SSL) protocol and the X.509 certificates. SSL uses public key cryptography to establish connections between client and server. Therefore each component of the UNICORE infrastructure has a public-private key pair with the public part known by the others. The keys have to be certified by a certification authority.

The user is authenticated by the gateway when presenting certificate. The user certificate is translated to the user login under which all tasks are executed on the target system. The user can use number of certificates to access different remote resources. The authentication scheme supports also project concept and jobs can be executed using different projects with single login. The UNICORE client provides users with tools for certificate management such as certificate signature request generation, keystroke editor or public key export.

The important advantage of the UNICORE architecture is an uniform access to the resources, independent of the underlying middleware. Each Virtual Site (Vsite) which is a single system or a cluster which shares the same user ids and file space is controlled by the Network Job Supervisor (NJS). It maps the UNICORE user id to the local user id at the target system, extracts and translates the jobs into real batch jobs, sends job-groups to be executed at the other sites to the corresponding gateway, provides local resource information to the gateway and takes care of the file transfer.

Site specific settings such as different paths to the particular application, local or system specific settings are introduced to the UNICORE through the Incarnation DataBase (IDB) entries which describe details of the local installation.

The UNICORE infrastructure allows also for registration of the applications available on the target machine through `SOFTWARE_RESOURCE` entries in the IDB. The UNICORE client has build-in capabilities to check software resource entries on the particular target systems and use it for job construction. Described

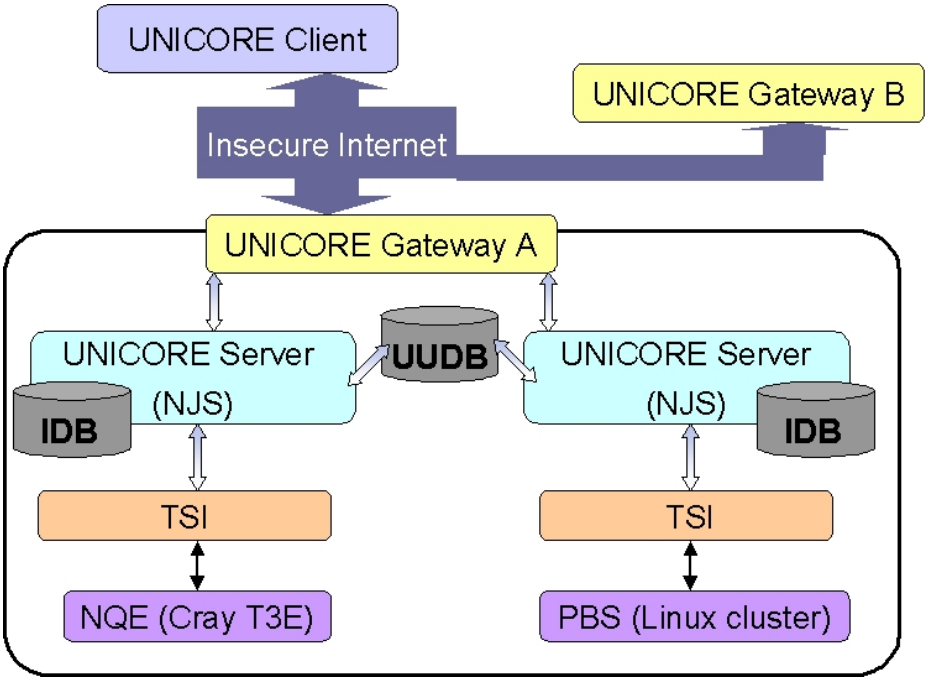


Fig. 1. The scheme of the UNICORE system. IDB denotes Incarnation DataBase, TSI - Target System Infrastructure and UUDB - UNICORE User DataBase

mechanism is especially useful for the pre-installed applications maintained by the system administrators.

The UNICORE has been adopted to the number of the target systems queuing software such as PBS, NQE, LSF, Load Level and Sun Grid Engine. Implementation to the other systems is relatively easily and requires only modifications of the target system infrastructure. User can also use UNICORE Client to submit job to the target system which provides resources with the Globus Toolkit.

The UNICORE client provides the plugin mechanism which becomes very attractive and efficient method for integration of applications with grid middleware. The plugin is written in Java and is loaded to the UNICORE client during start, or on request. Once it is available in the client, in addition to the standard futures such as preparation of the script job, user gets access to the menu which allows preparation of the application specific input and job files.

UNICORE fits to the Open Grid Software Architecture (OGSA) and provides interoperability with other existing grid middleware, in particular Globus Toolkit.

3 UNICORE Workflow Management

The UNICORE middleware has build in advanced workflow management. The user can build job in the form of an oriented acyclic graph, which covers majority of the user cases. Each node in the graph is represented by a single task (command task, script task or plugin) or a subgroup. Since the workflow build in the form of pure acyclic graph has some limitations, the UNICORE allows for conditional execution and loops. The conditional execution allows to switch to the different execution tree based on the logical conditions. The condition can be of different kind, for example return code value, test if particular file exists and has particular permissions. The decision can be made also based on the actual timestamp. Loop execution allows for sequential execution of the group of tasks given number of times or as long as execution conditions, similar to one used in the conditional execution, will be satisfy. User can control loop execution and within loop has access to the shell variables such as iteration index or loop length and user can use them for the parametric execution.

The UNICORE workflow management is relatively simple and intuitive and covers most important job scenarios. Significant advantage of this solution is integrated workflow editor which allows for graphical construction of the workflow. The UNICORE is oriented on the job execution, therefore data transfer associated with the workflow must be designed by the user. This possibility is especially important in the life sciences applications because user has to deal with number of different formats and data types and the only existing solution is utilization of filters.

The created workflow can be stored as Abstract Job Object (AJO) which is used as internal UNICORE format, or as the XML file. In the later case no standard job description schema is available and UNICORE specific tags are used. The standardized tag set could be used once it will be proposed.

The most important advantage of the UNICORE workflow model is possibility to use applications or groups of applications as single node in the workflow graph. The number of existing plugins makes such node easy to create and modify. User can also define required software resources for each node separately, as well as choose different target system for execution.

Number of AJO templates have been developed for most popular biomolecular applications including Gaussian98 [10], Gromos96 [11], Amber [12] and Charmm [13]. The templates can be easily modified by the user in the part including application input, exactly as it is done for the batch jobs. Significant difference comes from the fact using UNICORE middleware scripts can be run on any target system without modifications taking advantage of the environment variables defined in the IDB. The job configuration includes information on the input files which has to be transferred form the user workstation to the system where particular application will be executed as well as information on the output files which must be transferred back. Once this information is provided by the user all required file transfers will be performed automatically during job submission or retrieval of the results.

Currently, in addition to the standard script and command tasks, user can use plugins for job preparation for the most popular quantum chemistry codes such as Gaussian, Carr-Parrinello Molecular Dynamics [14], molecular dynamics codes such as Amber and Gromos or for the sequence comparison code BLAST [9, 15].

The dedicated plugin for monitoring of the execution of the running job is available. The plugin prepares service job which based on the provided UNICORE job number enters this job working directory on the target system and gets access to the generated files. Selected files can be retrieved to the users workstation as they exists in the working directory or can be filtered to transfer only the most significant information. This plugin provides user with typical filters such as retrieval of the last lines of selected file, but more advanced filters can be written. The postprocessed file is transferred to the UNICORE client and can be displayed in a text window or used for the visualization.

We have also developed extended visualization plugins which can be used for post processing including 2D and 3D visualization of the molecular data. [16]. The range of applications integrated with the UNICORE environment has been extended by us with access to the databases both available through web interface or accessed directly. The another developed extension is access to the remote experimental facilities. Both solutions are described below.

4 Database Access

An access to the remote databases plays important role in the life sciences. Currently most of the databases provide web interfaces to query and submit data. The good example of such databases is protein structural database - PDB [17] and sequence database ENTREZ at NCBI [18]. We have develop interfaces to these database in the form of the UNICORE plugin. This allowed us to integrate database search with the UNICORE environment and use them as part of the workflow. While plugin is started user can build database query using dedicated interface: simple, based on the keyword search, or advanced one with full functionality provided by the database.

The query is submitted to the databases web interface and result is returned to the UNICORE client. The search results can be browsed, visualized or saved for later processing. Since copies of both databases are available at different locations, the plugin checks automatically which mirror is currently available and for the user query the one which provides answer in the shortest time is used. We have decided to use existing visualization packages because users are used to them and there is no need to develop just another visualization tool. As result we were able to integrate UNICORE plugins with packages such as JMol, JMV or RasMol/RasWin as well as any other application available on the Client workstation.

Currently the HTTP protocol is used to communicate with the database server, but it can be replaced by more advanced one such as web services.

The another communication scenario is implemented by the DBAccess Plugin which allows for access to the SQL databases from the UNICORE Client. The

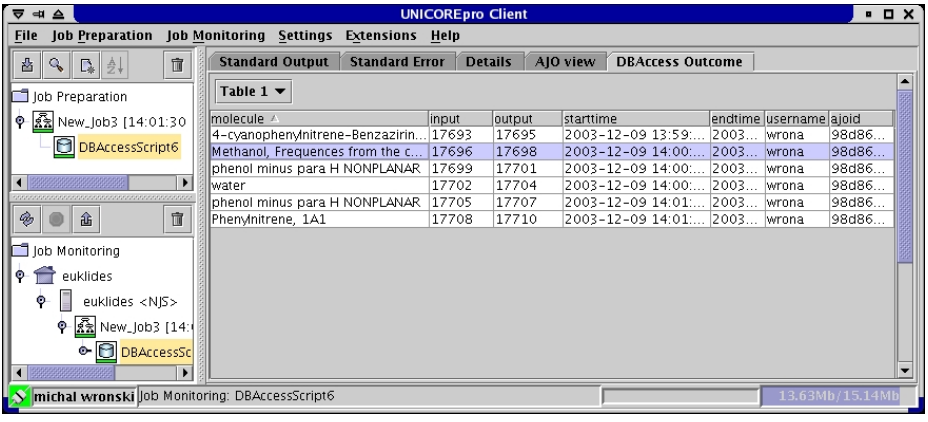


Fig. 2. The DBAccess plugin loaded into UNICORE Client. The outcome of the database query job is presented

access is realized with the help of the dedicated scripts run directly on the database serwer. The Unicore middleware provides secure methods for the access authorization and user authentication therefore the server can be separated from the internet by the firewalls and not directly accessible. Currently the DBAccess Plugin it is able to connect to PostgreSQL and MySQL - the most popular free databases. The interface to another database can be easily developed and added to the existing plugin.

With the help of DBAccess Plugin user can prepare database query in the dedicated window and than submit it to the target system. The results are retrived and displayed in the UNICORE Client window in the text form or using XML (see. Fig. 2). In the later case the user obtains number of tools to order and search results within table. The plugin can be also used to store data in the database and can be used as part of the job workflow.

The DBAccess Plugin offers also a possibility to communicate with SDSC Storage Resource Broker [19] providing user-friendly interface. In an easy way user can specify host, port, database, query etc. Query results can be stored as XML for later processing. Queries can be predefined in the Incarnation Database (IDB) as well as in the plugin defaults file. User can use queries prepared by the Vsite administrator while advanced user may use his own query to retrieve data form the database. In particular case the perl bindings to the SRB has been developed, but other mechanisms like developed recently SRB web services interface can be used.

5 Access to the Remote Equipment

The resources available on the Grid are not limited to the computational ones. As it was stated before, life sciences research uses large amount of data obtained

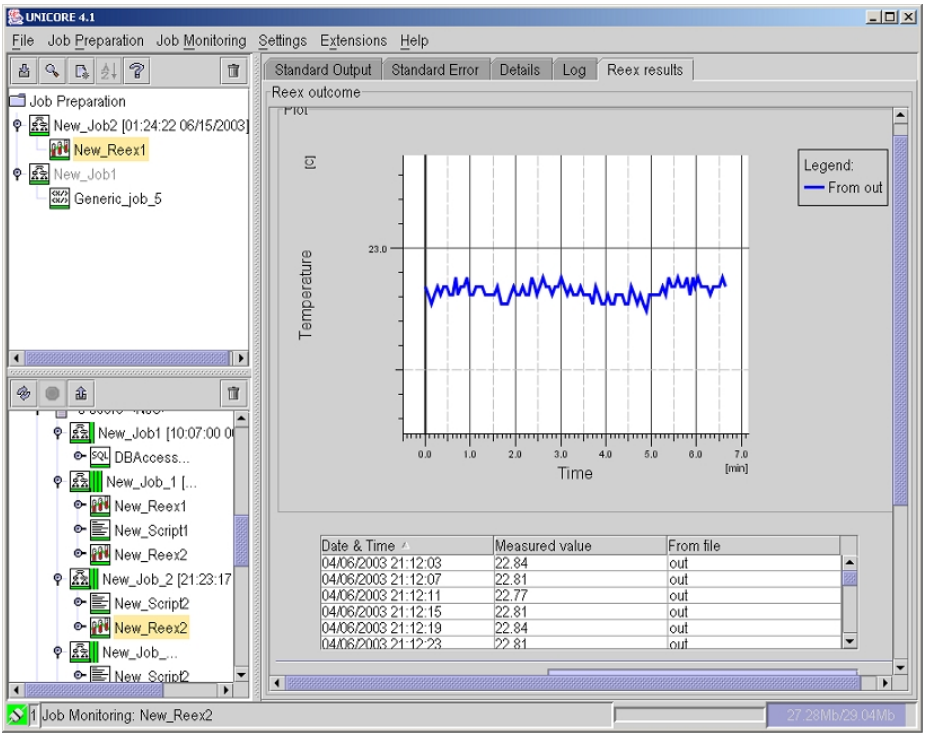


Fig. 3. An output from the measurement plugin

from the various experimental facilities. Therefore an integration of the various data sources with the computational and database systems is important.

To solve this problem, we have used UNICORE middleware to access remote experimental setups. In this case the interactive and semi-interactive access to the resources must be provided. Using dedicated extensions to the UNICORE Client user can control remote device and set up experimental conditions. The user can also use dedicated plugin to set up experiment and submit it to the target site as part of the job, in the exactly the same way as computational task. User also benefits from the UNCIORE authentication and authorization which secures access to the resources.

The results can be retrieved to the client workstation and postprocessed using visualization plugins within UNICORE Client or using other applications available at the client workstation. The described above plugins can be used to store results automatically in the database for further processing.

Developed plugin benefits from the interactive access functionality in the UNICORE middleware. The equipment control application is installed at the target system and it is registered in the IDB. The plugin constructs jobs for this application and retrieves output as presented in the Fig. 3. The plugin technology allows for immediate integration of the access to the measurement devices

with the users workflow and combine measurements with the simulations or data processing. The UNICORE allows also for interactive access to the remote instruments which can be used for the experiment set up and various adjustments.

This technology has been used to access devices connected to the target systems with serial or ethernet port, but interface to the other communication mechanism and protocols can be easily developed.

6 Conclusions

The UNICORE software was used as framework providing uniform access to the number of resources and applications important for life sciences such as computational chemistry and molecular biology. This includes seamless access to the computational resources, databases as well as experimental devices. The UNICORE software was used to establish European computational grid - EUROGRID [9, 20]. BioGRID is application oriented grid which adopt EUROGRID infrastructure to the specific area, namely molecular biology and quantum chemistry. Examples presented here demonstrates capabilities of the UNICORE middleware and shows directions for further development. Modular architecture based on the plugin concept and demonstrated visualization capabilities open number of possible applications in the different areas.

One should note, that UNICORE middleware can be used also together with other grid middleware, especially can be used as job preparation and submission tool for Globus.

Acknowledgements. This work is supported by European Commission under IST grants EUROGRID (IST-1999-20247), GRIP (IST-2001-32257) and UniGrids (No. 004279). The finacial support of State Comeetee for Scientific Research is also acknowledged.

References

1. Kesselman, C., Foster, I., *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufman Publishers, USA, 1999.
2. Foster, I., Kesselman, C., "Globus: A metacomputing infrastructure toolkit," *Int. J. Scientific Applications. 11, 2* pp. 115–128, 1997.
3. Legion. University of Virginia. Charlottesville. VA USA Http://legion.virginia.edu.
4. LSF. Platform Computing. http://www.platform.com.
5. Bhandarkar, M., Budescu, G., Humphrey, W. F., Izaguirre, J. A., Izrailev, S., Kal, L. V., Kosztin, D., Molnar, F., Phillips, L. C., Schulten, K., "Biocore: A collaboratory for structural biology," in *Proceedings of the SCS International Conference on Web-Based Modeling and Simulation*. (Bruzzone, A. G., Uchrmacher, A., Page, E. H. ed.) San Francisco, California, 1999.
6. Talia, D., "The Open Grid Services Architecture: where the grid meets the Web," *IEEE Internet Computing, 6, 6* pp. 67–71, 2002.
7. DataGrid WP10 (HealthGrid) http://edg-wp10.healthgrid.org/
8. UNICORE. Unicore Forum. http://www.unicore.org.

9. Bała, P., Lesyng, B. Erwin, D., EUROGRID - European Computational Grid Testbed. *J. Parallel and Distrib. Comp. 63, 5* pp. 590–596, 2003
10. M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, V. G. Zakrzewski, J. A. Montgomery, Jr., R. E. Stratmann, J. C. Burant, S. Dapprich, J. M. Millam, A. D. Daniels, K. N. Kudin, M. C. Strain, O. Farkas, J. Tomasi, V. Barone, M. Cossi, R. Cammi, B. Mennucci, C. Pomelli, C. Adamo, S. Clifford, J. Ochterski, G. A. Petersson, P. Y. Ayala, Q. Cui, K. Morokuma, P. Salvador, J. J. Dannenberg, D. K. Malick, A. D. Rabuck, K. Raghavachari, J. B. Foresman, J. Cioslowski, J. V. Ortiz, A. G. Baboul, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. Gomperts, R. L. Martin, D. J. Fox, T. Keith, M. A. Al-Laham, C. Y. Peng, A. Nanayakkara, M. Challacombe, P. M. W. Gill, B. Johnson, W. Chen, M. W. Wong, J. L. Andres, C. Gonzalez, M. Head-Gordon, E. S. Replogle, and J. A. Pople. Gaussian 98. 2001.
11. Van Gunsteren, W., Berendsen, H. J. C., *GROMOS (Groningen Molecular Simulation Computer Program Package).* Biomos, Laboratory of Physical Chemistry, ETH Zentrum, Zurich, 1996.
12. Kollman, P., *AMBER (Assisted Model Building with Energy Refinement).* University of California, San Francisco, USA, 2001.
13. Brooks, B. R., Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S., Karplus. M., "A program for macromolecular energy, minimization, and dynamics calculations," *J. Comp. Chem. 4*, pp. 187–217, 1983.
14. CPMD consortium. http://www.cpmd.org.
15. Wypychowski J., Pytliński J., Skorwider Ł., Benedyczak K., Wroński M., Bała Life Sciences Grid in EUROGRID and GRIP projects. *New Generation Computing 22, 2*, pp. 147–156, 2004
16. Bała P., Benedyczak K., Nowiński A., Nowiński K. S., Wypychowski J. Interactive Visualization for the UNICORE Grid Environment In: *ICCS2004*(Lecture Notes in Computer Science 1332) Eds. M. Bubak, G. D. v. Albada, P. M. A. Sloot, J. Dongarra, J. Springer-Verlag 2004 pp. 101-108
17. Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., Bourne. P. E., "The Protein Data Bank," *Nucleic Acids Research. 28* pp. 235-242, 2000.
18. http://www.ncbi.nlm.nih.gov/entrez
19. M. Wan, A. Rajasekar, R. Moore, P. Andrews A simple mass storage system for the SRB data grid. *Proceedings 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies.* IEEE Comput. Soc. 2003, pp.20-5. Los Alamitos, CA, USA
20. Pytliński, J., Skorwider, Ł., Bała, P., Nazaruk, M., Alessandrini, V., Girou, D., Grasseau, G., Erwin, D., Mallmann, D., MacLaren, J., Brooke, J., "BioGRID - An European grid for molecular biology," *Proceedings 11th IEEE International Symposium on Distributed Computig*, IEEE Comput. Soc. 2002, p. 412
21. Pytliński, J.,Skorwider, Ł., Huber, V., Bednarczyk, K., Bała, P., "UNICORE - An Uniform Platform for Chemistry on the Grid," *Journal of Computational Methods in Science and Engineering., 2, 3s-4s* pp. 369–376, 2002

MyGridFTP: A Zero-Deployment GridFTP Client Using the .NET Framework

Arumugam Paventhan and Kenji Takeda

School of Engineering Sciences, University of Southampton,
Highfield, Southampton, SO17 1BJ, UK
{povs, ktakeda}@soton.ac.uk

Abstract. Large-scale scientific and engineering applications are increasingly being hosted as Grid services using Globus middleware complying to the Open Grid Services Architecture (OGSA) framework. In order for users to fully embrace Grid applications, seamless access to Grid services is required. In working towards this aim we present the design and implementation of Grid clients that utilise the language-independent Microsoft .NET Framework that can be deployed without software prerequisites (zero-deployment). We demonstrate runtime security authentication interoperability between Microsoft Windows-native SSPI and the Globus GSSAPI, with full proxy support. This is demonstrated with a .NET GridFTP client, called MyGridFTP. We believe that this is one of the first implementations to use Windows native security infrastructure to interoperate with the Grid Security Infrastructure in Globus. This paves the way for language-independent .NET clients to be written that are fully interoperable with Globus-based Grid services. This work is part of a larger experimental aerodynamics Wind Tunnel Grid project, which has significant requirements for data management from acquisition, collating, processing, analysis and visualisation.

1 Introduction

The data acquired in scientific and engineering experiments must be stored, processed, and visualized effectively in order for users to be able to analyze the results in a coherent fashion. The resources utilized in these steps are often distributed across the network, at both inter-organization and intra-organization level. Grid technology can help build a coordinated resource sharing environment [1] in this scenario.

Grid clients developed using client side Application Programming Interface (API) allow application users to consume Grid Services. [2] demonstrates how the Java commodity technology can be used to access Grid services and envisages Commodity Grid (CoG) Kit on other platforms. There are two possible approaches to developing Grid clients:*Client-side application and Browser or Web-based application.* The advantage of client side applications is a rich user experience and access to the local file system. The disadvantage is that the necessary software and runtime environment must be installed and configured before

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 374–383, 2005.

it can start on the client machine. Any version change at the server side could affect the client, often requiring installation of a new version of the client. The advantage of Web-based applications is that the up-to-date version can be invoked without having to worry about any installation steps on the part of user. The disadvantages are that it will have no access to the local file system - a problem, for instance, with client initiated file transfers and accessing a local X.509 certificate store for GSSAPI authentication - and it is not configurable to connect to an arbitrary *IP Address/Port number*. For example, GridFTP utilizes port 2811, supports certificate based authentication and may be running on a server other than the Web server front-ending the Grid service. Also, in the case of third party GridFTP transfers the client needs to make control channel connections to multiple servers.

The .NET framework allows another possible approach to the above problem when we assume Windows client [3]. Since the .NET runtime environment and SSPI are integral part of Windows, a .NET based Grid client can be downloaded over HTTP similar to HTML document and run as a client-side application (Fig.1). In this way, it does not require client-side installation or configuration steps combining the advantages of both the client-side application and web-based application. The first time download is cached on the client machine for subsequent invocations and can be configured to only download again when there is a version change. The technology is comparable to Java Web Start. The advantages of zero-deployment using .NET include the ability to leverage rich client capabilities such as highly-interactive user interface; a simple user interaction does not have to take a round-trip for response; access to local filesystem for data transfer services; and ability to make network connections to any server.

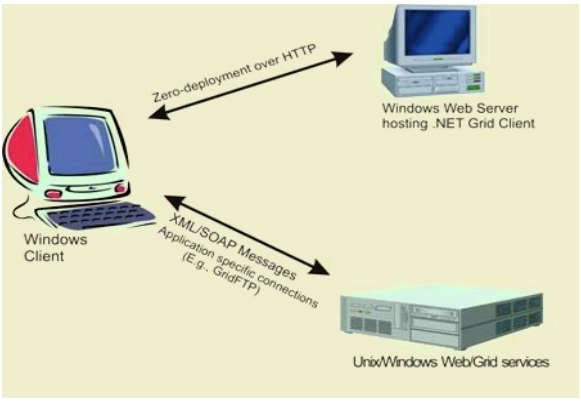


Fig. 1. Zero-deployment architecture

This work is part of a project to develop a Wind Tunnel Grid system aims at consolidating the workflow process for aerodynamics test engineers and scientists in test planning, data acquisition, processing, analysis, visualization, data management and data mining. The raw data from the Wind Tunnel experiments are

stored as flat files and its associated metadata are stored in Relational Database Management System (RDBMS). Authorized users are allowed to access the functionalities exposed as a set of Grid/Web services.

In the following sections we enlist the Wind Tunnel Grid requirements in general and .NET managed GridFTP client MyGridFTP in particular and describe its implementation details.

2 Wind Tunnel Experiments and Grid-Specific Requirements

The School of Engineering Sciences at the University of Southampton has a number of small and large wind tunnel facilities that are used for teaching, research and industrial applications, such as Formula One racing car, aircraft and high-performance yacht design.

A variety of measurement systems, such as particle image velocimetry (PIV), pressure transducers, microphones, video, digital photographs for flow visualisation and laser doppler (LDA) and hot-wire anemometry (HWA) are used. These require a disparate array of software, running on many standalone hardware systems. The data acquired using these systems varies in format, volume and processing requirements. The raw data is in a number of different flat file formats, typically proprietary binary formats and text files. Processing the data is performed using commercial software, mostly provided by the data acquisition hardware vendor, spreadsheets, user-written C and FORTRAN programs and Matlab scripts. Issues that arise from this arrangement include training, support, data compatibility, data access and analysis, and backup.

In order to improve the overall workflow efficiency a Wind Tunnel Grid system is being developed (Fig.2). This will allow experimental data to be managed,

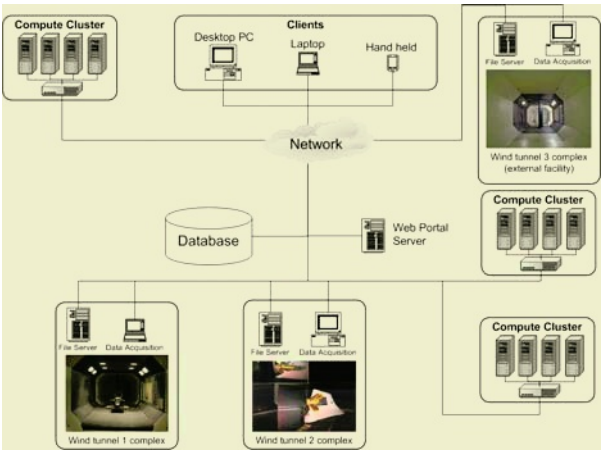


Fig. 2. Wind Tunnel Grid

stored and analysed in an RDBMS. For a given project users spread across multiple sites may perform complementary experiments, and use each other's compute resources for data processing and analysis. In order to provide seamless user access to the system, easy to use, lightweight client software is a major requirement. Typical users of the system include experienced aerodynamics engineers and scientists, undergraduate and postgraduate students, and visiting staff. Extensibility, so that additional data acquisition systems and analysis algorithms/software can be integrated into the system, is also required.

As a first step, the system should provide Globus GridFTP service to users for their data management. Data generated through wind tunnel experiments can be managed via the Wind Tunnel Grid portal from any .NET-enabled laptop or desktop on the network, without any client-side installations.

3 MyGridFTP Requirements

The major requirement of MyGridFTP is to provide GridFTP file transfer features including X.509 certificate based GSSAPI authentication. User should be able to invoke MyGridFTP from the Wind Tunnel Grid portal. This gives users the flexibility of transferring data files from any network location using their laptop or any desktop without having to install any software. MyGridFTP must also provide APIs for the development of custom clients supporting zero-deployment. Users should have options to configure GridFTP features - parallelism, striping, TCP Buffer size etc. From the project specific metadata available as part of the Wind Tunnel Grid system MyGridFTP can provide the user with an application specific automatic upload option. It means that the user does not have to select individual files for transfer, rather they simply need to select the folder and files are selected for transfer automatically. The graphical user interface must allow remote directory browsing of user's project space.

4 MyGridFTP Implementation

MyGridFTP is implemented using the Microsoft .NET runtime environment, as it enables application deployment from a Web location and rich client capabilites. It consists of GridFTP client APIs, a security module and graphical user interfaces. GridFTP client APIs include calls supporting features that are part of GridFTP extensions [4]. The security module uses SSPI and CryptoAPI[1], respectively to mutually authenticate with the GridFTP server and generate proxy certificates. MyGridFTP is hosted as part of the data transfer services in Wind Tunnel Grid Portal. The Wind Tunnel Grid Portal allows users to log into the system, create projects, sub-projects and test-cases, and upload raw data files from experiments. When the user clicks on the MyGridFTP link the .NET executable is automatically downloaded and run on the local machine supporting

[1] Microsoft Cryptographic Service Provider functions.

GSSAPI authentication, Globus Proxies and GridFTP data transfer features. The following sections elaborate the implementation with technical details.

4.1 Security

Grid security is crucial for authentication, authorization and delegation. The Generic Security Services Application Programming Interface (GSSAPI) [5] defines a portable API for client-server authentication. At the server side, Grid Security Infrastructure (GSI) is implemented using GSSAPI and Grid forum recommended extensions [6]. The GSI Message specification [7] defines three types of GSI messages exchanged between client and server as shown in Table 1. During Context-establishment, the MyGridFTP client uses SSPI to exchange SSLv3 handshake messages for mutual authentication with the GridFTP server. The client can delegate its credentials to server by sending a delegation flag. The server responds by sending a PKCS10 certificate request containing a valid public key and proxy certificate extension. A new proxy certificate with full/limited proxy credential is created, DER encoded and signed using CryptoAPI based on the Proxy Certificate Profile [8]. The Windows environment provides each user a personal *"MY"* store to manage user's X.509 certificates.

4.2 Runtime Environment

The .NET Framework consists of Common Language Runtime (CLR) and Framework Class Library (FCL). The CLR operates on assemblies which is a logical grouping of one or more managed modules or resource files. The *assembly* is defined as the smallest unit of reuse, versioning and security. Assemblies can consist of types implemented in different programming languages. The CLR is discussed and compared with the Java Virtual Machine (JVM) for multi-language support in [9]. The .NET development environment compiles high-level language code into Intermediate Language (IL). The CLR's JIT (just-in-time) compiler converts IL into CPU instructions at runtime. More details on architecture of .NET platform, about managed and unmanaged code can be found in [10].

MyGridFTP consists of three assemblies: the security module written in C^{++} for GSSAPI authentication and delegation of user credentials; GridFTP

Table 1. MyGridFTP Mutual authentication, Delegation and Message security

GSI Message Phase	MyGridFTP Client	SSLv3 Message Type	GridFTP Server
Context Establishment	AcquireCredentialHandle InitializeSecurityContext called until 'Finished' (implying successful authentication)	 ⇔Client-Server Hello ⇐ Server Certificate ⇐ Certificate Request ⇒ Client Certificate ⇒ Client Key Exchange ⇒ Certificate Verify ⇔ ChangeCipherSpec ⇔ Finished	gss_acquire_cred gss_accept_sec_context called until 'Finished' (implying successful authentication)
Delegation	⇒ Delegation Flag ⇐ PKCS10 certificate request ⇒ Proxy Certificate Chain	⇒ ApplicationData ⇐ ApplicationData ⇒ ApplicationData	gss_accept_delegation
Application-specific	EncryptMessage DecryptMessage	⇒ ApplicationData ⇐ ApplicationData	gss_unwrap gss_wrap

⇔ Both client and server exchange, ⇐ Server to client message, ⇒ Client to server message

client classes written in C#; and graphical user interfaces written C#. The security assembly uses interoperability services available in the .NET framework to invoke services available as part of SSPI and CryptoAPI. SSPI (Secur32.dll) and CryptoAPI (Crypt32.dll) are currently available as unmanaged implementations in Windows, but it is envisaged that these will be available as fully managed implementations in due course. Interoperability services available in the *System.Runtime.InteropServices* namespace in FCL expose mechanisms for managed code to call out to unmanaged functions contained in Dynamic Link Libraries (DLL). The user requires to make either the MyGridFTP assembly or the MyGridFTP download site as *trusted* by using .NET Framework configuration wizard, so that MyGridFTP client will have read/write access to the local file system.

4.3 Web Server Description

The Wind Tunnel Grid Portal consists of set of Active Server Pages (ASP.NET) web forms (HTML pages) and associated processing logic known as *code-behind files* written in C#. It is hosted using Internet Information Services (IIS) under Microsoft Windows Server 2003. The code-behind files instantiate the data access layer classes for accessing project metadata stored in the RDBMS (SQL server 2000). Some of the Wind Tunnel Grid metadata tables include *UserAccounts*, *Projects* and *TestCases*.

The application specific *Testcases* metadata vary depending on the Wind Tunnel experiment. The *Testcases* ASP.NET web page has the link for MyGridFTP, as shown in Fig.3. When the user clicks on the link the MyGridFTP executable is downloaded and runs on the client machine for GridFTP file transfer. Since the number and names of files to be transferred can be determined based on the *Testcases* metadata available, users can opt for an automatic data transfer option. In auto mode the files are transferred automatically; based on the direction of data transfer the user would select an upload folder or download folder. The user experience is seamless, in that they are not aware that a separate rich client application has been downloaded, installed and run.

4.4 GridFTP Server Configurations

The GridFTP server component part of Globus Toolkit 2.4 is configured on a Linux platform. It runs as an *inetd* network service on port 2811. Since the UserAccounts metadata table holds the home directory information, the system administrator has the flexibility of mapping multiple Wind Tunnel Grid users to the same unix login based on projects or user roles. The /etc/grid-security/grid-mapfile below shows a sample authorization entries.

"O=Grid/OU=GlobusTest/OU=simpleCA-cedc10.eng.soton.ac.uk/OU=eng.soton.ac.uk/CN=Paventhan" wtg

"O=Grid/OU=GlobusTest/OU=simpleCA-cedc10.eng.soton.ac.uk/OU=eng.soton.ac.uk/CN=Kenji Takeda" wtg

"O=Grid/OU=GlobusTest/OU=simpleCA-cedc10.eng.soton.ac.uk/OU=eng.soton.ac.uk/CN=C Williams" wtgadmin

In this example the first two users are authorized to login as 'wtg' (Wind Tunnel Grid User) and the last as 'wtgadmin' (Wind Tunnel Grid Administra-

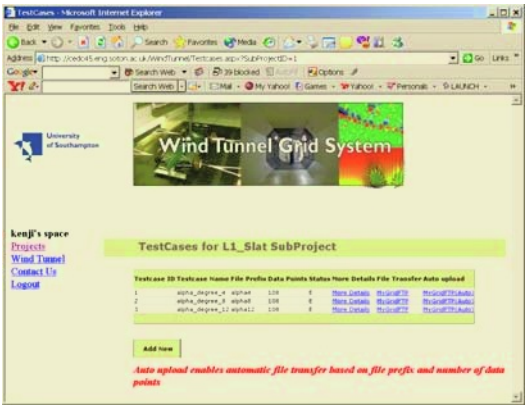


Fig. 3. Webpage hosting MyGridFTP

tor). UserAccounts table holds the actual home directory metadata, which could be, for example, a subdirectory in /home/wtg for the first two users - this gives them different project space. MyGridFTP makes an XML Web service request immediately after authentication and sets the home directory metadata as the FTP root directory during file transfer.

4.5 User Interfaces and Features

Fig.4. shows the user interface when invoked from the MyGridFTP(Auto) http link in Fig.3. Once downloaded MyGridFTP reads the user certificate store, authenticates with the GridFTP server, delegates user credential and performs an auto upload from selected folder. The auto upload feature is based on *Testcases* metadata (see section 4.3). In case of the manual option, the user needs to select individual files for transfer.

MyGridFTP can be configured for parallelism in extended block mode during Upload/Download. At the API level the MyGridFTP class (Table 2) supports GridFTP protocol for extended retrieve (ERET), extended store (ESTO) and striped data transfers (SPAS or SPOR). The ExtendedStore and ExtendedRetrieve allow partial copy (data reduction) of the file to be transferred. Partial

Table 2. MyGridFTP Class and its usage

Class: MyGridFTP **Functions:**	**Usage**
Authenticate(x509Cert.ThumbPrint) ParallelUpload(localFile, remoteFile, nDataPaths) ParallelDownload(remoteFile, localFile, nDataPaths) ExtendedStore(localFile, remoteFile, offset, length) ExtendedRetrieve(remoteFile, localFile, offset, length) Mode(modeString) Type(typeString) Upload(localFile, remoteFile) Download(remoteFile, localFile) List(listParams) ... and other basic FTP calls	*// read user's personal certificate store here* *mygftp = new MyGridFTP(ipaddress, port);* *mygftp.Delegation = true;* *mygrftp.DelegationType = LimitedProxy;* *mygftp.Authenticate(userX509Cert.ThumbPrint);* *mygftp.Mode(MyGridFTP.ExtendedMode);* *mygftp.ParallelUpload(localFile, remoteFile, 2)*

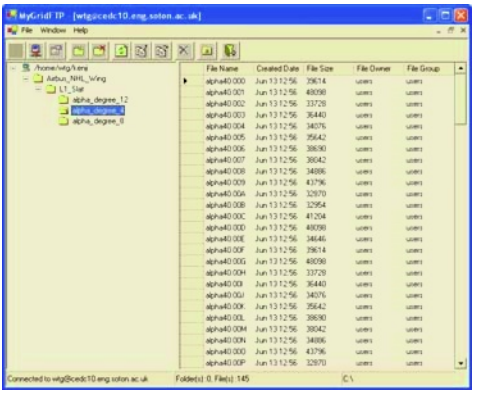
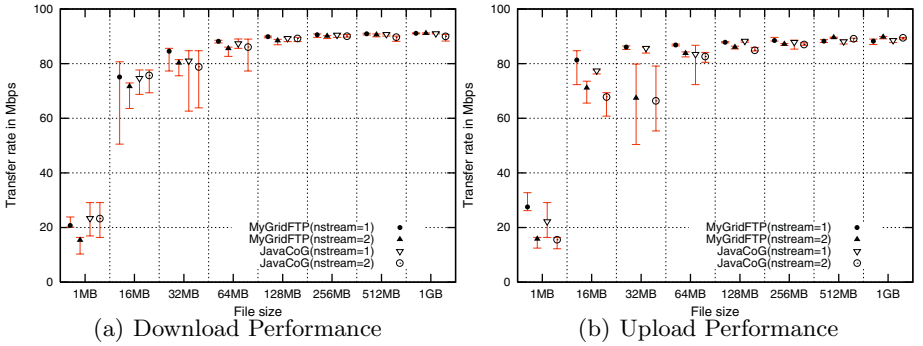


Fig. 4. MyGridFTP graphical user interface

data transfer will be more useful when we implement processing and visualization components of the Wind Tunnel Grid portal. Similarly striping and third party transfers are programmatically possible using the MyGridFTP class.

4.6 Performance

MyGridFTP download and upload performance are compared with Java CoG GridFTP client for various file sizes as shown in Fig.5. The performance test was measured within two separate programs written in C# and Java, respectively, using MyGridFTP APIs and Java CoG APIs. The GridFTP clients were run on a Intel Pentium-4 2.2 GHz Desktop running Windows XP and GridFTP server was configured on a Dual Intel Pentium-III 450 MHz, Linux system. The client and server are connected over a 100 Mbps Ethernet LAN. The time taken for Grid server authentication and authorization was measured separately over number of runs to compare Grid security implementations of MyGridFTP and



(a) Download Performance (b) Upload Performance

Fig. 5. MyGridFTP Performance

Java CoG as shown in Table 3. The X.509 user certificate for this experiment is of 1024 bit key length. The lesser authentication time of MyGridFTP could be attributed to its use of native runtime and security infrastructure. Each file was transferred between client and server a number of times and minimum, average and maximum bandwidths were recorded. As can be seen from the plot, large file sizes have very less bandwidth variability between different runs. Also, parallel streams does not improve the performance for smaller files as the test were run in a LAN environment. Both Java CoG and MyGridFTP measures a maximum download performance of 91 Mbps and upload performance of 89 Mbps for 1GB file size. As a comparison, iperf [11] bandwidth measurement tool gives 92.5 Mbps for 1 GB file input. By default, Windows XP operating system sets the TCP send and receive buffer size to 8K. This value is limiting, especially, for file transfers over wide area networks (WANs). The TCP buffer size could be increased in MyGridFTP programmatically to suit to high bandwidth-delay product subject to Operating Systems upper limits.

Table 3. GSSAPI Authentication

Authentication Time	Minimum	Maximum	Average
JavaCoG	1891ms	2563ms	2122ms
MyGridFTP	734ms	1156ms	848ms

5 Discussion

As the .NET Common Language Infrastructure (CLI) has been ratified as an ECMA standard (ISO/IEC 23271) there is interest in implementations on non-Windows platforms. For example, the Mono project [12] is an open source implementation of the .NET framework for use on Linux, Unix and Windows. Mono enables .NET managed code to run on multiple platforms similar to Java. The Mono.Security.Protocol.Tls namespace part of mono .NET project implements a 100% managed Transport Layer Security (TLSv1.0) and SSLv3. Incorporating GSSAPI authentication using this namespace will make MyGridFTP (and other clients requiring X.509 certificate based authentication) run on multiple platforms. Hence, the approach to Grid client deployment by means of hosting and executing on multiple platforms using the .NET framework will become realizable. Also, the Microsoft .NET Compact Framework targeting mobile devices will enhance the Grid application reach to Pocket PCs, PDAs and Smart Phones.

The Globus Reliable File Transfer (RFT) service could be hosted on Wind tunnel data acquisition system, data management server and compute cluster. This would allow client initiated asynchronous data transfer and notifications. If user is interested in uploading or downloading the data to their local system at any stage of the workflow, client/server style interactive data transfer is required.

The Wind Tunnel Grid portal could selectively implement XML Web services-based approach and GSI-security based approach for services such as GridFTP

for maximum compatability with other Grid resources. As future work, the MyGridFTP API can be extended to include GridFTP protocol improvements [13], resource management and information management making it a full-blown Globus client kit based on .NET.

6 Conclusions

A zero-deployment GridFTP client using Windows native runtime (.NET) and Security infrastructure (SSPI and CryptoAPI) is described. MyGridFTP launches from a web location and runs on the client machine without any software prerequisite, it can access the local file system and allows server-to-server communications via proxy certificates. We believe that this is one of the first Grid client implementations to use the Windows native security infrastructure (SSPI) to interoperate with the Grid Security Infrastructure (GSSAPI) in Globus, and as such it provides the basis for a language-independent .NET-based Commodity Grid (CoG) Kit.

References

1. Foster I, Kesselman C (eds.):The Grid: Blueprint for a Future Computing Infrastructure, Morgan-Kaufmann (1999)
2. Gregor von Laszewski et.al: A Java commodity grid kit, Concurrency and Computation: Practice and Experience, vol. 13 (2001) 643-662
3. Duncan Mackenzie:Introducing Client Application Deployment with *ClickOnce*, Microsoft Developer Network (2003)
4. W. Allock (ed.):GridFTP: Protocol Extensions to FTP for the Grid, Global Grid Forum Recommended Document (2003)
5. Linn J.:Generic Security Service Application Program Interface, Version 2, Update 1, RFC 2743, (2000)
6. Meder S., et al:GSS-API Extensions, Global Grid Forum Document (2002)
7. Welch Von (ed.):Grid Security Infrastructure Message Specification (2004)
8. Tuecke S., et.al:Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, IETF (2004) `http://www.ietf.org/rfc/rfc3820.txt`
9. Erik Meijer:Technical Overview of the Common Language Runtime, Microsoft Research, Technical Report (2001)
10. Jeffrey Richter:Applied Microsoft .NET Programming, Microsoft Press (2002)
11. http://dast.nlanr.net/Projects/Iperf/
12. http://www.go-mono.com
13. I.Mandrichenko (ed.):GridFTP v2 Protocol Description, Global Grid Forum Recommended Document (2004)

On Using Jini and JXTA in Lightweight Grids

Kurt Vanmechelen and Jan Broeckhove

Department of Mathematics and Computer Sciences,
University of Antwerp, BE-2020 Antwerp, Belgium
kurt.vanmechelen@ua.ac.be

Abstract. Many production level Grid systems deployed today are characterized by their focus on the integration of dedicated high-end systems and their high administration and maintenance costs. We make the case for lightweight Grid middleware as a prerequisite to harnessing general computing resources in order to achieve large scale computing. We outline the characteristics of the environment in which such a lightweight Grid operates and the ensuing middleware requirements. We analyze the possible contribution of two distributed frameworks, Jini and JXTA, in achieving these requirements.

1 Introduction

Grid systems have gained tremendous popularity in recent times because they hold a promise of enabling secure, coordinated, resource sharing across multiple administrative domains, networks, and institutions. Compute Grids typically consist of a relatively small and static group of, permanently and completely dedicated, high-end nodes. These nodes are the supercomputers and clusters which can be found in the participating research centers. Also, the Grid resources are typically not open in the sense that anyone can submit jobs to them. Usually, they are set up to solve only one particular set of problems. This model, known as heavyweight Grids (HWG), has been adopted in several projects and software toolkits, such as Globus, the Enabling Grids for E-science project in Europe (EGEE) and many others. Overviews can be found in [1] and [2].

Many applications of smaller magnitude do not require a heavy software stack delivering explicit coordination and centralized services for authentication, registration, and resource brokering as is the case in traditional Grid-systems. For these applications, a lightweight and stateless model, in which individuals and organizations share their superfluous resources on a peer-to-peer basis, is more suitable. A potential role for Jini [3] and JXTA [4] technologies in such a model has been recognized in a recent e-Science Gap Analysis report [5]. In this contribution, we will consider the requirements for lightweight Grid middleware and review the contributions of the Jini and JXTA frameworks in this regard.

2 LWG Environmental Characteristics

In contrast to their heavyweight counterparts, lightweight Grids (LWG) broaden the scope of resource sharing by including lower-end computing resources as

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 384–393, 2005.

found in desktop machines or laptops at home or in the office. This results in a new and significantly different environment in which the lightweight Grid has to operate. In this section we discuss the properties of this environment and the resulting consequences on the requirements for the Grid middleware.

Limited Administrative Resources. Heavyweight Grids mainly integrate supercomputers and high-end clusters that are hosted in an institutional context. These systems are maintained and administered by dedicated professional personnel. This lowers the 'ease of administration' requirement on heavyweight Grids. Often heavyweight Grids have an important installation and maintenance cost, even for relatively simple applications demands [6]. However, because of the small number of system administrators involved, it is feasible to provide training for the administrators and reduce the burden on end-users.

In the lightweight Grid setting, a large number of people will want to share their resources. However, these individuals will not have the technical expertise nor the time to administer their resources on a day to day basis. Therefore the middleware should be highly self configurable, self maintainable and very easy to install and administer.

In addition to the larger number of users, a higher number of applications will be deployed on a global lightweight Grid, each with its own requirements concerning the execution environment. Therefore, special attention should be given to the ease of deployment of new applications in the Grid and automated preparation of the execution environment hosted by the computational resources in support for these applications.

Heterogeneity. Another consequence of including general desktop resources is the higher degree of both hardware and software heterogeneity in the Grid fabric. At the software level, the middleware needs to deal with differences in operating systems, installed libraries, local security policies, etc. At the hardware level great variances related to CPU, storage, memory and bandwidth availability have to be dealt with. In heavyweight Grids homogeneity of the software backplane is sometimes dictated by the strict adherence to OS and middleware package listings that are managed centrally (e.g. EGEE). This is no longer possible in lightweight Grids as resources are expected to be non-dedicated.

Volatility. In clusters or heavyweight Grids, the resources are permanently and completely dedicated to the Grid. This is no longer the case for home and office resources. The resource owner has to be able to enforce full control on the conditions under which resources are shared. These include the requirement of non-obtrusiveness (i.e. only superfluous resources are shared), time based requirements (e.g. not during office hours) or virtual organization (VO) specific requirements (support for a limited and dynamic set of applications and VO's). This results in a highly volatile Grid fabric and strongly varying resource availability. This volatility has a prominent impact on the middleware components that make up the lightweight Grid. Firstly, because resources leave and join the

Grid fabric at high rates, efficient and dynamic lookup, discovery and monitoring of resources becomes more important. Secondly, more complex scheduling policies and algorithms are required. Finally, fault tolerant execution of composite workflows (*failure transparency*) and migration of jobs or services between resources (*migration transparency*) become more important. The same goes for having a persistent service state (*persistence transparency*) and for support of atomic transactions (*transaction transparency*).

Scale. Due to the large number of participants in a lightweight Grid, both as resource providers and resource consumers, scalability is of the utmost importance. With large numbers of concurrent nodes, all forms of centralized services have to be approached with caution. Services such as lookup, discovery, resource brokering, scheduling, accounting and monitoring have to be implemented in a highly scalable and distributed fashion. On the other hand, lightweight Grids also have a role to play in setting up relatively small, institutional Grids. Such Grids have different requirements for the middleware's services. Consequently, a general purpose lightweight Grid middleware should be highly *customizable* and *modular*, making it possible to selectively assemble components depending on the deployment context.

Openness. Heavyweight Grids are typically closed systems in that only a limited number of parties are authorized to submit jobs. The right to consume resources in a heavyweight Grid setting is linked to membership within a particular VO. Enrollment in a VO however, is still dependent on human intervention for establishing the identity of the prospective member. Also, the knowledge resource providers have of the applications that a VO hosts is limited, as is the knowledge resource consumers have of the characteristics and quality of the resources that are generally available. All of this makes joining a Grid and formulating service level agreements for resource sharing, a less than transparent proposition.

The even higher number of users and applications targeted by a lightweight Grid will require a more dynamic and automated form of resource sharing. Service level agreements and collaborations will need to be established on the fly based on the *system's knowledge* of resource provider and consumer characteristics and policies. This requires the necessary accounting and monitoring infrastructure to be in place.

More importantly, in this open peer-to-peer setting, some incentive or reward has to be given to resource providers. This could be achieved by a token based approach. Providers earn tokens for sharing their resources and are able to trade them for remote resources later on. For resource providers that are not consumers, a conversion from tokens to another (hard) currency or service is necessary. A first step in this direction has been taken by the Compute Power Market project and the Nimrod/G resource broker [7].

Hostility. A lightweight Grid should be able to operate in an environment where a multitude of independent clients consume the resources delivered by a mul-

titude of independent providers across the internet. This is in effect a hostile environment where resource consumers, owners and their process of information exchange should be protected from malicious actors. Protection of the provider system against malicious consumers or third-parties becomes an important issue. If LWG middleware needs to be easy to install, a strict, but easy to understand, security model should be adopted. Proper authentication and authorization mechanisms should be in place in order to be able to trace user actions and limit their potentially negative consequences.

At present, lightweight Grid systems do not address all of the issues outlined above, often focusing on one particular aspect among the environmental characteristics. According to the taxonomy given in [5] an LWG operating in the above environment would be characterized as an Autonomic and Lightweight Peer-to-Peer Desktop Grid. The following sections will discuss how Jini and JXTA can contribute to such a Grid.

3 An Overview of Jini and JXTA

Clearly the design and implementation of a lightweight Grid system is an extensive undertaking. Leveraging existing technologies or technology components is indispensable. A number of groups have considered Jini [8, 9, 10, 11] or JXTA [12, 13, 11] in the realization of their Grid middleware. Both technologies, developed by Sun Microsystems, have a mission statement that involves enabling or facilitating distributed computing in a dynamic environment.

3.1 Jini

The design of the Jini framework and its resulting programming model are strongly influenced by the recognition of the *Eight Fallacies of Distributed Computing* [14]. Jini's goal is to enable the creation of virtually administration free service communities. It offers the developer the possibility to partition a user community and its resources into groups, also called federations, and to publish and discover services within the context of a federation. To this extent, *lookup services* (LUS) take on the brokerage of *service proxies* within federation(s). Services can be hardware or software oriented and are expected to join and leave the network in a non-orderly fashion. In the Jini vision, clients interact with services through a representative called a proxy, which is a Java object that is shipped to the client's JVM in order to mediate all client-server interactions. Clients of the discovery infrastructure find lookup services by means of multicast or unicast protocols [3]. Services can be looked up in the LUS with type based object-oriented matching semantics based on the proxy's type, or with attribute value matching semantics based on the the attribute values accompanying the proxy.

For client-to-service interactions through the proxy, Jini provides a pluggable service invocation layer called JERI (Jini Extensible Remote method Invocation) with RMI based semantics. JERI exposes the layers of the service invocation process to the developer. Three layers are defined, the *invocation* layer, the *object*

identification layer and the *transport* layer. As a result, new policies and implementations for object marshalling, distributed garbage collection, wire protocols, and method dispatching can be plugged in.

Jini offers a distributed programming model that includes dynamic lookup and discovery, support for distributed events, transactions, resource leasing [15] and a security model. The main features of the security model are strong support for safe delivery of mobile code, mutual client-server authentication and authorization, and secure client-server communication. The leasing model contributes to the construction of self healing systems with low administration costs by automatically reclaiming service resources upon client failure. Acquisition of a (remote) resource includes negotiating a lease period during which this resource will remain allocated to the client. After this period, the resource owner is free to reclaim the resource. A client may of course renew the lease.

3.2 JXTA

The JXTA framework consists of the specification of six XML protocols that enable the creation of dynamic user communities in a heterogenous peer-to-peer environment. The framework aims to support key peer-to-peer (P2P) abstractions such as multi-peer communication, peers and peer groups [4]. A *pipe* abstraction is used to represent end-to-end communications between peers over a *virtual overlay network* [16]. This virtual network enables seamless communication between peers in the presence of networking barriers such as firewalls and Network Address Translation (NAT), through the use of *relay* peers.

All resources in the JXTA network are described by XML documents called advertisements. Standard advertisements defined by the JXTA framework include, advertisements for peers, peer groups, pipes and services. JXTA presents general application logic as a resource in the form of module advertisements. Three different classes of module advertisements are defined. *Module Class advertisements* embody a certain behavior, but do not contain the implementation for that behavior nor the means to appeal to it. They can be used to group a set of *Module Specification advertisements* who embody the access specification to the module. Often such a module specification advertisement contains a pipe advertisement that can be used to set up a communication channel with the service. *Module Implementation advertisements* contain the physical form of the platform specific application logic for the service that implements a particular module specification.

JXTA supports client-server interaction models through *peer services* as well as more collaborative models through *peer group services*. One invokes a peer service through a contact point embedded in the service's module specification advertisement. Although client to service communication is left open ended in the JXTA framework, one should use JXTA pipes for transmitting messages between peers in order to benefit from the global connectivity features of the JXTA overlay network. Messages are constructed in XML and are converted to a binary wire format for efficient transfer over the network. In the group service model, a client does not have to interact with a representative through its contact

point but can appeal directly to the service network by posting a request that requires collaboration among different service instances. The key service within the JXTA framework that enables this form of service interaction, is the *resolver* service. The resolver enables nodes to automatically distribute generic queries among service instances and receive the associated responses.

Security-wise, JXTA offers secure, TLS compliant, pipe communications [17]. In JXTA, the peer group may be used as a security context. Access to the peer group, and by consequence to all services advertised in the group, can be guarded by a group membership service. Such a membership service may require a peer to authenticate itself before access to the group and its services is granted. Apart from the membership service, a JXTA peer group also hosts standard slots for components responsible for advertisement discovery, query propagation, routing and peer monitoring. As such, every peer group may define its own infrastructure components and access to these components is limited to group members. Groups can also be hierarchically structured, a standard top level group called the *World* group delivers the minimal infrastructure to discover and become part of other subgroups.

4 Contributions of Jini and JXTA to an LWG Middleware

In this section we discuss a mapping of the features outlined the previous section, to the middleware requirements of a lightweight Grid as discussed in section two.

4.1 Limited Administrative Resources

The creation of *virtually administration free* service communities is one of the key mission statements of the Jini framework. The most notable contributions of Jini in this regard are:

- Proxy autonomy: The use of proxies allows the service designer to include client side logic that automates certain tasks and eases the amount of administration forced upon the user. An example is to include a user interface that is automatically deployed on the client machine.
- Low cost service updates: services are easy to upgrade. As soon as a new version of the service proxy is uploaded and the service provider discontinues the lease on the old proxy, client side discovery will return the new service proxy. As code is automatically downloaded to the client, no manual steps are involved to upgrade the client side proxy code.
- Dynamic registration/deregistration: When services go live they automatically register with the discovery infrastructure. When the service's hosting environment goes down the registration lease is discontinued and all stale information is automatically cleared from the discovery infrastructure.
- Low cost middleware installation: Java based middleware can be installed across the Web with a single click through the use of Java Webstart or custom applets.

– Leasing Model: The leasing model as a whole contributes to the development of autonomous self healing services that are able to recover from crashes and clear stale information without administrator intervention.

4.2 Heterogeneity

As both Jini and JXTA have Java based implementations, the Java features for dealing with OS and hardware heterogeneity are inherited by both frameworks. These features include uniform access to file system and network, code and data portability and a uniform security framework. We note however that a pure Java based approach is not mandated by Jini nor JXTA. While Jini requires a Java platform on the client side to cope with system heterogeneity, a service's backend can be implemented in any language on any platform. JXTA on the other hand, aims to provide different implementations of its framework for different languages and systems. These implementations are able to communicate with each other because of the framework's standardized XML protocols. Consequently, both frameworks are able to integrate non-Java services in the Grid middleware. Also, support for heterogeneous networks is one of JXTA's key goals. The presence of relays in the network is transparently used by the JXTA routing protocols to provide communication links between peers across protocol and firewall boundaries. Despite the frameworks' contributions, significant heterogeneity issues still need to be dealt with by the middleware itself. These include the dependencies Grid applications will have on a particular execution environment, extension of scheduling algorithms to cope with widely varying system characteristics and the consequences of site autonomy in respect to local security policies.

4.3 Volatility

The idea that distributed resources are volatile is inherently addressed in the Jini philosophy. Features such as dynamic registration and deregistration of services in the discovery architecture with proper client notification, support for two phase commit transactions, a leasing model, and the presence of sequence numbers in the eventing system reflect this. In Jini, every client interacts with a service's backend through a mediator called a proxy. The use of smart proxies enables service developers to include fault recovery policies in the proxy in order to shield clients from the consequences of a service going down. A Jini proxy thus contributes to the fulfillment of the *failure transparency* requirement. Jini services also integrate well with the RMI activation framework which enables the automatic recovery of services in the event of the hosting JVM going down and coming back up again. This fulfills the *persistence transparency* requirement. Jini's strong support for mobile code and mediation of communication channels through smart proxies, contribute to the *migration transparency* that is necessary to adapt to changing fabric conditions in a volatile environment. Jini deals with the *transaction transparency* requirement by offering standard services for setting up two phase commit transactions.

JXTA's *group service* model automatically distributes queries among all service instances in the group, and thus provides a fault tolerant way of accessing service functionality. As long as one service instance is up, the query will be answered. JXTA also features a fault tolerant message routing infrastructure. Upon detection of a failed link a new route is automatically resolved for the message. JXTA uses a distributed hash table (DHT) [18] approach to provide efficient discovery of resources and avoid flooding the network with discovery queries. However, maintaining the consistency of a DHT under volatile conditions is costly. JXTA combines a loosely consistent DHT with limited range walks around the network to deal with this trade-off. It mitigates the cost of keeping a DHT consistent under high volatility rates and can deal with varying levels of node volatility.

4.4 Scale

Jini has firm roots in object-oriented distributed programming. Its modular architectural style and programming model contribute to the development of extensible middleware that supports customized deployment depending on the scale of deployment. This is demonstrated by the Jini ICENI project [11]. Jini lookup services have been found to scale well as the number of registered services increases [19]. Scalability studies of the same extent are not available for JXTA although performance studies of the framework on a smaller deployment scale are available [20]. Jini lacks support for discovering computational resources on a global scale in a fully distributed manner, but JXTA does have this capability. Integrating Jini and JXTA discovery infrastructures and plugging in JXTA pipes into the JERI protocol stack is a possible course of action in this regard. Other schemes for extending Jini's discovery reach exist [10][21]. Ubiquitous large scale discovery and communication is one of JXTA's key goals. A fully distributed discovery infrastructure based on a hierarchical peer-to-peer network, and the ability to assign infrastructure resources such as routing, discovery and membership services to specific peer groups, contribute to the fulfillment of the scalability requirement. In principle, JXTA's peer-to-peer infrastructure is better qualified to address scalability requirements by avoiding central server bottlenecks on a network and system load level. However, there have been practical problems in the past regarding the scalability of JXTA's implementation. Although the 2.0 release addressed scalability issues [16], more research remains necessary [20].

4.5 Openness

Jini and JXTA do not provide features that have a direct impact on the openness of the resulting LWG middleware. As mentioned in section 2, making a computing platform openly accessible involves creating incentive for resource owners to share and consumers to consume, which is beyond the scope of both frameworks. However, JXTA's pluggable approach to group membership services and the ability of every peer to start, join or leave a peer group, does support the creation of more dynamic and open communities than currently found in

heavyweight Grid VOs. Jini's support for mobile code is beneficial to the development of an agent based framework that is able to implement a dynamic form of service level agreement (SLA) negotiation between providers and consumers. In this regard, group communication in a JXTA peer group can also help in implementing economic pricing and bidding models.

4.6 Hostility

Both Jini and JXTA offer means to protect information exchange in a hostile environment. Jini provides TLS/SSL and Kerberos GSS-API based communicators for client service interactions, while JXTA provides TLS pipes for secure communication. Jini also has strong support for safe delivery of mobile code.

Mutual client-server authentication and authorization schemes integrated within the Jini framework, allow involved parties to establish identity, trust and grant each other the right access permissions. Once parties trust each other enforcement of access rights is further controlled by the security provisions delivered by the Java platform. This includes sandboxing the mobile code to limit its range of impact on the hosting system. Although this scheme allows for safe execution of pure Java code, it has no support for safe execution of non-Java binaries. Apart from TLS transports, JXTA also includes standardized slots for group membership services that control peer access to the peer group in question, a feature missing in Jini. The implementation of this group service is up to the peer group creator. For lightweight Grid deployments, a group membership model that is based on peer reviewing [22] would be an interesting option for certain virtual organizations.

5 Conclusion

In this contribution we have looked at the key characteristics of the environment in which a lightweight Grid has to operate and how this determines requirements for lightweight Grid middleware. We have presented a technical overview of Jini and JXTA in order to analyze their possible contributions to the fulfillment of these requirements. Although neither platform delivers a silver bullet to the Grid middleware developer, we have identified areas in which both platforms clearly show their contributions. As such, both technologies have their own role in constructing a lightweight Grid middleware that will provide a more open and dynamic base for deploying Grids in a hostile, volatile and heterogeneous environment with limited administrative resources.

References

1. Ian Foster, Carl Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure* (Morgan Kaufmann, 2003).
2. Fran Berman, Geoffrey Fox, Anthony J.G. Hey, *Grid Computing: Making The Global Infrastructure a Reality* (John Wiley & Sons, 2003).

3. K. Edwards: *Core Jini* (Prentice Hall, 1999).
4. D. Brookshier, D. Govoni, N. Krishnan, J.C. Soto: *JXTA : Java P2P Programming* (Indiana, Sams Publishing, 2002).
5. G.Fox, D. Walker: e-Science Gap Analysis. http://www.nesc.ac.uk/technical_papers/UKeS-2003-01/GapAnalysis30June03.pdf
6. J. Chin, P. Coveney: Towards tractable toolkits for the Grid: a plea for lightweight, usable middleware. UK e-Science Technical Report, number UKeS-2004-01, 2004
7. D. Abramson, R. Buyya, J. Giddy: A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*, 18(8), 2002, pp. 1061-1074.
8. Y. Huang: JISGA: A Jini-Based Service-Oriented Grid Architecture, *International Journal of High Performance Computing Applications*, 17(3), 2003, pp. 317-327.
9. Y.M. Teo and X.B. Wang: ALiCE: A Scalable Runtime Infrastructure for High Performance Grid Computing. *Proceedings of IFIP International Conference on Network and Parallel Computing*, Springer-Verlag Lecture Notes in Computer Science Series 3222, Xian, China, 2004, pp. 101-109.
10. Z. Juhasz, A. Andics, K. Kuntner Szabolcs Pota: Towards a Robust and Fault-Tolerant Multicast Discovery Architecture for Global Computing Grids. *Proceedings of the 4th DAPSYS workshop*, Linz, Austria, September 2002, pp. 74-81.
11. N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington: Implementations of a Service-Oriented Architecture on top of Jini, JXTA and OGSI. *In Second Across Grids Conference*, Nicosia, Cyprus, 2004
12. S. Majithia, M. Shields, I. Taylor and I. Wang: Triana: A Graphical Web Service Composition and Execution Toolkit. *Proceedings of the IEEE International Conference on Web Services (ICWS'2004)*, San Diego, California, USA, 2004, pp. 514-521.
13. T. Ping, G. Sodhy, C. Yong, F. Haron and R. Buyya, A Market-Based Scheduler for JXTA-Based Peer-to-Peer Computing System. *Springer-Verlag Lecture Notes in Computer Science Series 3046*, 2004, pp. 147-157.
14. P. Deutsch: The eight fallacies of distributed computing. http://today.java.net/jag/Fallacies.html
15. M. Kircher, P. Jain: Leasing pattern. *Proceedings of the Pattern Language of Programs conference*, Allerton Park, Monticello, Illinois, USA, 1996.
16. B. Traversat, A. Arora, M. Abdelaziz et al.: Project JXTA 2.0 Super-Peer Virtual Network. http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf.
17. Sun Microsystems: Security and Project JXTA, Sun Microsystems, Inc, 2002.
18. E. Pouyoul, B. Traversat, M. Abdelaziz: Project JXTA: A Loosely-Consistent DHT Rendezvous Walker. Sun Microsystems, Inc., 2003.
19. M. Kahn, C. Cicalese: CoABS Grid Scalability Experiments. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(1-2), 2003, pp. 171-178.
20. E. Halepovic, R. Deters: The Costs of Using JXTA. *Proceedings of the 3th IEEE International Conference on Peer-to-Peer Computing (P2P'03)*, Linköping, Sweden, September 2003, pp. 160-167.
21. W.-H. Tseng, H. Mei: Inter-Cluster Service Lookup Based on Jini. *Proceedings of the 17th IEEE International Conference on Advanced Information Networking and Applications*, Xian, China, March 2003, pp. 84-89.
22. S.D. Kamvar, M.T. Schlosser, H. Garcia-Molina: The EigenTrust Algorithm for Reputation Management in P2P Networks. *Proceedings of the 12th International World Wide Web Conference*, Budapest, Hungary, May 2003, pp. 640-651.

Ticket-Based Grid Services Architecture for Dynamic Virtual Organizations

Byung Joon Kim, Kyong Hoon Kim, Sung Je Hong, and Jong Kim

Department of Computer Science and Engineering,
Pohang University of Science and Technology (POSTECH),
San-31, Hyoja-dong, Pohang, KOREA
{caleb80, jysh, sjhong, jkim}@postech.ac.kr

Abstract. A Virtual Organization (VO) in the Grid is a collection of users and distributed resources, in which resources are shared by users. VOs are dynamically created for some goals and then disappear after the goals are achieved. Conventional Grid architectures have been proposed for a single or static VO environment. In this paper, we propose a ticket-based Grid services architecture to support the dynamic VO environment. Tickets contain information and delegated rights of resource providers, VO managers, and users. By using these tickets, the proposed architecture supports various services in the dynamic VO environment, such as VO management, fine-grained authorization, resource information, and resource management.

1 Introduction

Due to the rapid growth of technologies of computer and network, much research has focused on computing with a plenty of resources which are heterogeneous and scattered geographically. The Grid has started from the concern of scientific computation over geographically distributed systems and has been an emerging technology in recent years. The Grid is defined as a controlled and coordinated resource sharing and resource use in dynamic, scalable virtual organizations [1]. Many studies [2, 3, 4] have been conducted on resource information, resource management, security, and implementation of the Grid.

A Virtual Organization (VO) in the Grid consists of a set of users and resource providers under the spirit of sharing [1, 5]. Individuals who want to collaborate on a common goal make a VO in the Grid and share information through the VO. A user in the VO has easy access to resources in that VO, while resource holders utilize their resources more efficiently throughout the VO. Thus, a VO generally consists of users, resource providers, and VO managers.

As the Grid computing spreads rapidly in recent years, the need for dynamic VOs is apprehended. VOs for various purposes are created and then disappear after their goals are achieved. In a VO, many users join and leave according to their needs. Resource providers can also dynamically change their policies in the VO, such as available time, provided resources, and so on. In the dynamic

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 394–403, 2005.

VO environment, each user and resource provider can participate in several VOs simultaneously. So, a resource provider can receive resource requests from users of several different VOs. Also, a user can request resources from several VOs. Although recent research [4, 6, 7] on fine-grained authorization has proposed in a single VO, few research has focused on the dynamic VO environment.

In this paper, we present a ticket-based Grid (Ti-Grid) services architecture for dynamic virtual organizations. The proposed architecture is based on tickets issued by VO members. Tickets contain information and delegated rights of resource providers, VO managers, and users. A VO is easily created and managed by the tickets published by resource providers and the VO manager. Each user knows the VO information and acquires the necessary tickets for his jobs. Since only the rights specified in the tickets are permitted, fine-grained authorization is also supported. The Ti-Grid architecture uses distributed agents to share and find tickets efficiently.

The remainder of this paper is organized as follows. The overview of the proposed architecture is provided in Section 2, which includes the components of the architecture, the description of tickets, and the ticket management system. Section 3 will describe the Grid services supported by the proposed architecture. In Section 4, we will explain the current implementation issue. Finally, this paper is summarized in Section 5.

2 Overview of Ti-Grid Architecture

2.1 Components

The Grid environment supporting for dynamic virtual organizations is composed of VO managers, resource providers, and users. A *VO manager* organizes a VO of resource providers and users who share a common goal of the VO. The VO manager knows which resource providers and users join the VO. Moreover, the VO manager keeps the VO's policy, such as the resource amount that each user can take. A *resource provider* grants its resource to users in the Grid, specially to those in the VOs that the resource provider joins. Resource providers need not know information about all users in the VOs, but ensure that only users in the VOs use their resources. A *user* is an end entity to run a job in the Grid. Users submit and run their jobs in the resource sites of their own joined VOs.

The proposed architecture uses tickets to provide the dynamic VO-enabled Grid services. A VO manager and resource providers in a VO issue their own tickets to organize the VO. Users in the VO issue tickets to submit and run their jobs in the VO. The entities that are geographically separated each other operate the VO by sharing these tickets.

In order to share and locate tickets from various entities, we use a distributed agent system for ticket management. In the proposed architecture, ticket management agents are implemented as a structured peer-to-peer system to provide an efficient ticket location service. VO managers and resource providers pub-

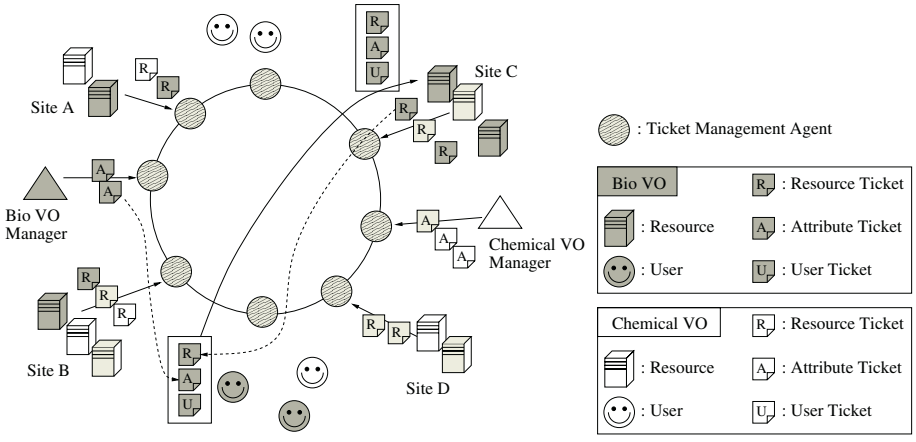


Fig. 1. An example of the Ti-Grid architecture

lish their tickets to the agent system. Users request the agent system to find appropriate tickets for their jobs.

Figure 1 depicts an example of the Ti-Grid architecture, in which four sites or resource providers are available, two VOs are organized, and several users want to use resource in the Grid. Bio VO's resource in Figure 1 consists of one from site A, one from site B, and two from site C. Likewise, Chemical VO is composed of resource from four sites. Resource providers issue and publish their resource tickets. Each VO manager also issues and publishes attribute tickets for VO users. The published tickets are registered to the distributed agent system. In Figure 1, a user in Bio VO obtains necessary tickets from the agent and submits a job with the tickets to site C.

2.2 Tickets

The main idea of the Ti-Grid architecture is to use tickets for the assertion of delegated rights in a virtual organization. A ticket is unforgeable and exchangeable among VO entities for resource control. A ticket record is an XML object asserting that the specified policy controls a resource over the specified time interval in the ticket. To protect the integrity of a ticket, each ticket record is signed with the private key of the issuer.

Tickets are classified as a resource ticket, an attribute ticket, a user ticket, and a job ticket. The former three tickets are signed by each issuer to prevent forgery. The *resource ticket* is issued by a resource provider and describes the provider's local policy for a VO. Resource providers issue various resource tickets for several services and VOs. A resource ticket includes ticket fields, such as Id, Issuer, VOName, ValidTime, UseTime, ServiceType, Condition, and so on. Id, Issuer, and VOName specify the unique ticket identifier, the resource provider's name, and the VO name, respectively. ValidTime corresponds to the ticket's

```
<ResourceTicket Id="2004HPC112">
    <Issuer>/O=Globus/O=Hpc/Ou=ticketgrid.com/CN=hpc2<Issuer>
    <VOName>/O=Globus/O=Hpc/Ou=ticketgrid.com/CN=VO1<VOName>
    <ValidTime start="2004−04−18−12−00" end="2004−04−18−13−00"/>
    <UseTime>01−00−00</UseTime>
    <ServiceType>
        <wsdl:service name="ManagedJobFactoryService">
         ......
        </wsdl:service>
        <Condition>
            <rslrestriction>
                <maxCpuTime>720</maxCpuTime>
                <maxMemory>300</maxMemory>
                <jobType>mpi</jobType>
                <jobType>single</jobType>
            </rslrestriction>
        </Condition>
    </ServiceType>
    <Signature Id="S1" xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo/>
    </Signature>
</ResourceTicket>
```

Fig. 2. A sample resource ticket

lifetime, so that the expired ticket is not useful anymore. UseTime means the guaranteed usable time of a job with the ticket, which will be explained in Section 3.2. ServiceType and Condition fields inform the provided service name and the resource provider's policy for the VO. The signature of the resource provider is also attached to the ticket.

Figure 2 shows a sample resource ticket. The name of the resource provider is '/O=Globus/O=Hpc/Ou= ticketgrid.com/CN=hpc2' and the joining VO is '/O=Globus/O=Hpc/Ou=ticketgrid.com/CN=VO1.' The expiration time of the ticket is '2004-04-18-13-00', as described in the ValidTime field in the ticket. The resource provider re-issues the resource ticket if the ticket is not used until the expiration time.

The *attribute ticket* is issued by a VO manager to assert the VO manager's policy for each user in the VO. VO managers specify attribute or policy of users in their VOs in this attribute ticket. Ticket fields in the attribute ticket are the same as those in the resource ticket, except that the UserName field indicates the user name. However, the fields in the attribute ticket mean the VO manager's policy for the user.

The *user ticket* is issued by a user who will submit a job to the resource site in a VO. The user ticket asserts the necessary rights to run the job. A user delegates his or her rights to the user ticket under the policies of the resource ticket and the attribute ticket. The user ticket also contains ticket fields similar to other tickets, such as Id, Issuer, UseTime, ServiceType, and Condition. Differently from other tickets, the user ticket has ImportedTicket field, which references the resource ticket and the attribute ticket.

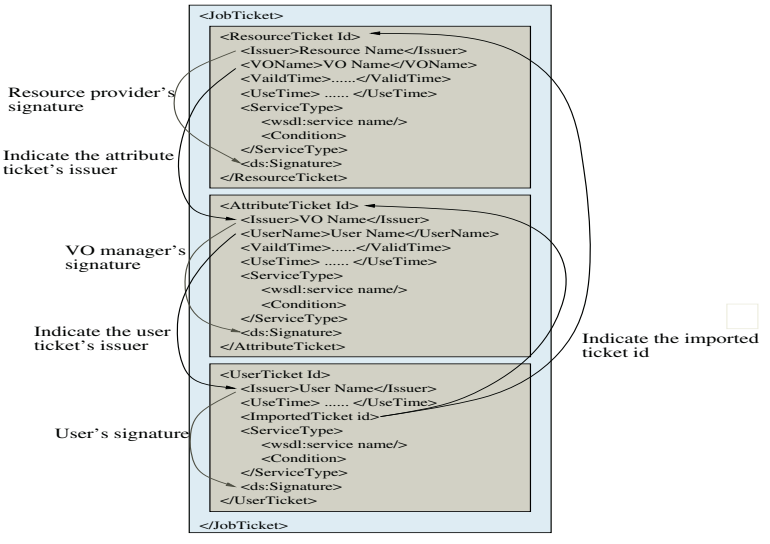


Fig. 3. An example of a job ticket

The *job ticket*[1] is generated by a user in order to request a job. A user concatenates the other three tickets and forms the job ticket, as shown in Figure 3. The ImportedTicket field in the user ticket indicates which resource and attribute tickets are included in the job ticket. The job ticket is used to authorize the submitted job.

2.3 Ticket Management Agent System

The ticket management in the proposed architecture is accomplished by distributed agents. For the cooperation between agents and the efficiency of the ticket management service, we construct the agent system with the peer-to-peer system, such as Chord [9]. Each agent keeps the registered tickets and shares the ticket information with other agents through the peer-to-peer system.

The ticket management service includes ticket registration, ticket location, and ticket revocation. Resource providers and VO managers publish and register their newly issued tickets to the agent system. When resource providers recognize that their resources are available to VOs, they issue tickets and register them to the agent system. VO managers also publish their users' attribute tickets when new VOs are created or policies of VOs are changed. These registered tickets are shared among the agents so as to make location service efficient.

[1] In our previous work [8], the job ticket was called the *TAS ticket*. TAS (Ticket-based Authorization Service) is the name of the authorization system in the proposed architecture.

When a user requests resource tickets or attribute tickets, the ticket location service of the agent system searches the location of requested tickets and returns the tickets. In addition, each agent removes the expired tickets in its ticket pool. All tickets have the ValidTime field which implies the expiration time of the ticket. Since the expired tickets are of no use, they are automatically removed from the agent system, which improves the performance of location service.

The distributed agent system may have a lot of routing overhead for ticket registration and location. To reduce this overhead and improve performance, we construct the distributed agent system with the Chord [9] which is a structured P2P architecture based on DHT (Distributed Hash Table). In the Chord, each node, which is the agent in our system, has a unique identifier ranging from 0 to 2^{m-1}. These identifiers are arranged in a circle, where each node maintains information about its successor and predecessor on the circle. Additionally, each node also maintains information about at most m other neighbors for efficient routing. Thus, data lookup takes $O(\log N)$ hops, where N is the total number of nodes in the system. It guarantees that all existing tickets matching a query will be found within the bounded costs in terms of the number of messages and involved agents.

3 Ticket-Based Grid Services Architecture

3.1 Dynamic VO Management

The Ti-Grid architecture provides the dynamic VO environment based on tickets. In the Grid, many VOs are created for various goals and disappear after accomplishing the goals. The VO policy also changes dynamically as the VO operates normally. The proposed tickets are used to construct this dynamic VO environment.

In order to create a new VO, resource providers and the VO manager publish their tickets. The resource providers in the VO issue their resource tickets that assert resource providers' policies for the VO. The VO manager also issues and publishes attribute tickets for users in the VO. After these tickets are registered to the agent system, the VO is created conceptually. Users who want to run their jobs in that VO can request the rights with proper resource tickets and attribute ticket.

When a VO need not operate any more, the VO manager and resource providers should not publish their tickets. No ticket for the VO in the agent system implies the disorganization of the VO.

The policy change of a resource provider or a VO manager is also easily realized with a newly issued ticket. When a resource provider wants to adjust the amount of resource to the VO, the resource provider reissues and publishes the resource tickets that reflect the policy about the changed amount. If a resource provider does not want to provide resource any more, it stops publishing resource tickets. The VO manager can also issue attribute tickets again if the policy for some users changes. Moreover, a new user can easily join the VO by obtaining a

new attribute ticket issued by the manager. On the other hand, a user is deleted from the VO as the VO manager stops issuing the user's attribute ticket.

3.2 Fine-Grained Authorization

When a user wants to run a job using the VO resource, the user requires resource tickets from the resource provider and an attribute ticket from the VO manager. The user can acquire these tickets from the distributed agent system and choose proper tickets for the job. When one resource ticket dose not have enough rights for the user's job, the user can use several resource tickets at the same time unless the total rights of the resource tickets do not exceed the rights asserted in the attribute ticket. After obtaining the necessary tickets, the user issues a user ticket that describes the needed rights for the job. The user generates a job ticket by concatenating the resource tickets, the attribute ticket, and the user ticket. As shown in Figure 3, the job ticket envelops the other tickets. The ImportedTicket field in the user ticket references the other resource tickets and the attribute ticket, which prevents modification of the ticket set. Since each included ticket is signed by the issuer, it cannot be forged by a malicious intermediate.

After the job ticket is ready, the user submits the job with the job ticket to the resource provider. The resource provider starts to verify the job ticket. The job ticket verification is performed in two steps, which are signature verification and content verification. Signature verification confirms that the job ticket is not forged. The signatures on the user ticket, the attribute ticket, and the resource tickets enveloped in the job ticket are verified by the public keys of each issuers. The resource provider also inspects whether Ids of the resource tickets and the attribute ticket are the same as those in Imported Ticket of the user ticket.

Next, the content verification process starts. Requested rights in the user ticket should be less than the issued rights in the resource tickets and the attribute ticket. Thus, the resource provider checks whether the requested rights in the user ticket are within the bounds of the total issued rights in the resource tickets and those of the attribute ticket. Then, the resource provider also confirms the availability of the requested service. After signature and content verification, the resource provider grants the requested rights to the user's job.

While the authorized job is running, the resource provider does not issue resource tickets for the used service. Resource tickets are reissued for other users only after the job is completed or the UseTime on the user ticket expires. Therefore, the user's job is guaranteed to use the resource for the UseTime period.

3.3 Resource Information and Management

The proposed architecture provides the VO information service, as well as the Grid information service. When a user submits a job to the Grid, the user should know which resource providers or sites have the available resource to run the job. Moreover, the information service in the dynamic VO environment should ensure that the provided resource information is classified according to each VO. The distributed agent system in the proposed architecture supports the VO

information service by gathering the VO attribute ticket and the resource tickets for the VO. For a given VO, the agent system provides the information of the available resource amount that each VO user can take.

Although users have the resource information, the submitted jobs are not guaranteed to be run. However, in the Ti-Grid architecture, the job tickets assure that the submitted jobs run in the resource sites. Resource providers offer their resource as long as the job ticket is valid. While executing the admitted job, they do not issue the resource tickets any more for the currently used resource. Thus, the resource tickets shared in the ticket management system are those for the currently available resource.

In addition, the accounting service can be easily supported in the proposed architecture. It is difficult to alter or forge tickets used in the job authorization. The enveloped tickets in the job ticket are signed by the resource provider, the VO manager, and the user, respectively. Besides the difficulty in modifying tickets, the fact that only the job with the job ticket is permitted to run enables more exact accounting for the job.

4 Implementation

The Ti-Grid architecture has been implemented to work on Globus toolkit 3 [5]. We implemented a ticket generator and the distributed agent system which shares tickets. And we also implemented the Gatekeeper service on the Globus toolkit. All components have been implemented in Java.

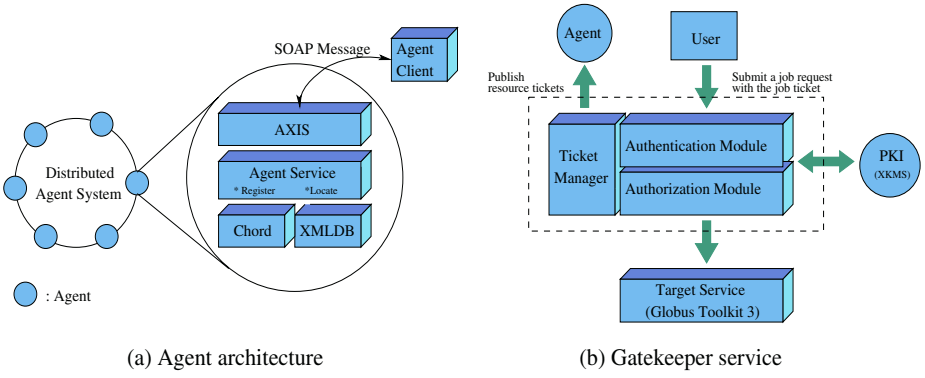
4.1 Agent

The agent shares and provides published tickets. We implemented the distributed agent based on the Chord which is a structured P2P architecture. The agent performs the ticket registration and location service. In the registration service, the agents save published tickets in the local XML database and advertise the ticket's reference to other agents. In the location service, the agent finds the advertised ticket reference and provides proper tickets to users.

The agent architecture is shown in Figure 4(a). Axis is essentially a SOAP engine that is a framework for constructing SOAP message service. Agent Service is a message style web service that receives and returns ticket XML documents in the SOAP envelope. XMLDB is designed to store and retrieve large numbers of XML documents such as tickets. Since Chord is based on a fast consistent hash function, it is necessary to define keywords to map agents. In the Ti-Grid architecture, we make the resource ticket's keyword by concatenating the VO name and the service name, and the attribute ticket's keyword by concatenating the user name and the VO name.

4.2 Gatekeeper Service

In the resource part, we implemented a Gatekeeper service and modified the Globus toolkit to recognize the Ti-Grid architecture. As shown in Figure 4(b),



(a) Agent architecture (b) Gatekeeper service

Fig. 4. Implementation of agent and Gatekeeper service

the Gatekeeper service is composed of three modules, which are the authentication module, the authorization module, and the ticket manager module. The authentication module enables to authenticate users. This module performs signature verification which provides integrity, message authentication, and signer authentication. For signature verification, authentication module communicates with XKMS [10] which is a XML based public key infrastructure. The authorization module provides ticket-based fine-grained authorization on the authenticated user. This module checks that the requested rights in the user ticket are within the bounds of the total issued rights in the resource ticket and attribute ticket.

After the authorization process, Gatekeeper service invokes the target service of Globus toolkit 3. The ticket manager module generates and publishes resource tickets. This module includes a ticket parsing module to create tickets and an XML signature module to perform the signature process. While the target service is running, this module does not generate and publish a new resource ticket for the used service. Only after the job is completed or the UseTime in the user ticket expires, it reissues and publishes new resource tickets for other users.

5 Summary

In this paper, we proposed the Ti-Grid architecture for dynamic virtual organizations. The proposed Ti-Grid architecture uses tickets that are unforgeable and exchangeable among VO entities for resource control. A VO manager, resource providers, and users in a VO issue their own tickets for delegating their rights. The VO is constructed by sharing these tickets. The VO also changes its policy as each Grid entity reissues the ticket containing the changed policy. Since the published tickets contain delegated rights of the issuers, fine-grained authorization is supported. The VO information service and VO resource management are also provided by using the tickets.

To share and locate tickets from various entities, we use a distributed agent system for ticket management. Agents in the distributed agent system share their tickets from the Grid entities. We use a structured P2P system for agents so that locating tickets is provided efficiently.

The previous implementation work has been focused on the fine-grained authorization service using tickets. Our current work includes the implementation of the VO information service and the further research on the accounting service in the proposed architecture.

Acknowledgment

This work was supported in part by the Ministry of Information and Communication of Korea under the Chung-Ang University HNRC-ITRC program supervised by the IITA, and the Ministry of Education of Korea under the BK21 program towards the Electrical and Computer Engineering Division at POSTECH.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. International Journal of Supercomputer Applications **15(3)** (2001) 200–222
2. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. International Journal of Supercomputer Applications and High Performance Computing **11(2)** (1997) 115–128
3. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. Proc. of 10th IEEE International Symposium on High-Performance Distributed Computing (2001) 181–194
4. Pearlman, L., Welch, V., Foster, I., Kesselman, C.: A community authorization service for group collaboration. Proc. of IEEE 3rd International Workshop on Policies for Distributed Systems and Networks (2002)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open Grid services architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum (2002)
6. Keahey, K., Welch, V., Lang, S., Liu, B., Meder, S.: Fine-grain authorization policies in the GRID: Design and implementation. Proc. of 1st International Workshop on Middleware for Grid Computing (2003)
7. Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Frohner, Á., Gianoli, A., Lõrentey, K., Spataro, F.: VOMS, an authorization system for virtual organizations. Proc. of European Across Grids Conference (2003) 33–40
8. Kim, B. J., Hong, S. J., Kim, J.: Ticket-based fine-grained authorization service in the dynamic VO environment. Proc. of ACM Workshop on Secure Web Services (2004)
9. Stoica, I., Morris, R., Nowell, D. L., Karger, D. R., Kaashoek, M. F., Dabek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup protocol for Internet applications. IEEE/ACM Transactions of Networking **11(1)** (2003) 17–32
10. XML Key Management Specification (XKMS). http://www.w3.org/2001/XKMS/

Heterogeneity of Computing Nodes for Grid Computing

Eamonn Kenny, Brian Coghlan, John Walsh, Stephen Childs,
David O'Callaghan, and Geoff Quigley

Department of Computer Science, Trinity College Dublin, Ireland
{ekenny, coghlan, john.walsh, stephen.childs, david.ocallaghan,
geoff.quigley}@cs.tcd.ie

Abstract. A heterogeneous implementation of the current LCG2/EGEE grid computing software is supported in the Grid-Ireland infrastructure. The porting and testing of the current software version of LCG2 is well under way on Red Hat 9, Fedora Core 2, IRIX and AIX. The issues concerned are discussed, and recommendations are presented for improvement of portability to non-reference operating systems.

1 Introduction

The computing needs of CERN's Large Hadron Collider (LHC) are to be met by the deployment of a grid service utilizing computing resources in many countries. The LHC Grid (LCG) project [1] is prototyping this infrastructure. Much of its current software, called LCG2, derives from the previous European DataGrid (EDG) project [2] that prototyped many of the necessary technologies. In similar fashion, the Enabling Grids for E-science in Europe (EGEE) project [3] is prototyping the future production grid for Europe, based on the LCG software. The CrossGrid project [4] is exploring near-interactive grid environments, first based on an EDG, now on a LCG, and soon on an EGEE foundation. Many other European grid projects are likewise building upon these foundations.

The LCG and EGEE software originally assumed reference ports to Red Hat 7.3 and Microsoft Windows, but subsequently this has been revised to include Scientific Linux 3 (SL3). Unfortunately this is a very restrictive situation that also contravenes the original heterogeneous ethos of grid computing. As a result of our interest in heterogeneity, we at Trinity College Dublin began porting to non-reference platforms in October 2003. Subsequently EGEE have almost finished porting the current LCG2 grid implementation to Scientific Linux on 32-bit and 64-bit architectures. Other work has been carried out by Casey[5] to port the replica management software to Darwin, on which Apple's Mac OS X is built. Further work has been begun by Maroney[6] at Imperial College London to convert some of the Red Hat 7.3 version of the LCG2 worker node code to Fedora Core 2.

For reasons discussed below, within the Grid-Ireland infrastructure we have been attempting to fully port some basic LCG2 worker node functionality to Red Hat 9, Fedora Core 2, IRIX 6.5, AIX 5.2L and Mac OS X v10.3, sufficient to

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 404–413, 2005.

perform Globus and EDG job submission. On this base we are attempting a full port of the Virtual Organisation Management System (VOMS), the Relational Grid Monitoring Architecture (R-GMA) and the Replica Management (RM) software. Since April 2004 we have been working closely with Maarten Litmaath, Carlos Osuna, Zdenek Sekera and James Casey in CERN and Vincenzo Ciaschini at INFN to achieve our goals.

Below we outline the motivations, the porting process itself, auto-building of the software, deployment and early benchmarking results.

2 Background and Motivation

Whilst the experimental and theoretical science paradigms remain strongly embedded in Irish science, there is strong growth in the hybrid paradigm, computational science. Most of this scientific computing is still done on local facilities. It involves a wide range of application areas, but few truly parallel applications. Most users develop their codes but use commercial libraries and tools. The reference architectures for these are a major factor in the choice of High-Performance Computing (HPC) architecture, i.e. most of the deployed architectures are mission-specific.

The HPC facilities in Ireland are very limited, an aggregate of about 1000 CPUs and 10TB of disk farms. There is funding for several medium-scale clusters (100-500 CPUs) and two medium-scale data farms in 2004/5.

Currently there is no large-scale HPC facility in Ireland. Until very recently there was no identified governmental intention to have one. In August 2004, however, the Irish government announced an initiative for the creation of a National Centre for High End Computing. It is very likely that the resulting centre will include a mix of mission-specific architectures.

These limited and mostly mission-specific resources should not be wasted by being inaccessible to the Grid simply because their architectures are not those of the reference ports.

There are four further considerations:

- Minimizing the demand on human resources by minimizing the proportion of the software that needs to be ported. The simplest component of most grid software frameworks is that relating to the worker nodes.
- Minimizing the demand on human resources by maximizing the proportion of the software that does *not* need to be ported. Thus all the non-worker node components should use the reference port, i.e. the core infrastructure should be homogeneous.
- Maximizing the availability of the infrastructure. Grid-Ireland has designed a transactional deployment system to achieve this[7], for which a homogeneous core infrastructure is highly desirable.
- There is interest in Irish computer science circles about issues of languages, programming models[8] and execution models[9][10] for heterogeneous environments, and Grid-Ireland specifically wishes to support these research directions.

Thus *Homogeneous Core Infrastructure, Heterogeneous Resources* is a pervasive motto that encapsulates explicit and implicit principles:

- *Explicit homogeneous core infrastructure:* this principle enables a uniform dedicated core national grid infrastructure, which supports a uniform architecture based on reference ports of the grid software, and thereby frees resources for maximum focus on the critical activities such as security and monitoring/information systems. Logically, it allows a uniform control of the grid infrastructure that guarantees uniform responses to management actions.
- *Implicit decoupling of grid infrastructure and site management:* this principle enables the infrastructure and sites to be independent. It can encompass policies, planning, design, deployment, management and administration. In particular it allows the infrastructure upgrade management to be independent of that of the site, and non-reference mission-specific architectures to be deployed at the site.

Grid-Ireland has been designed with this approach since mid-2001. Funding was sought and eventually granted, a senior Grid Manager appointed, a Grid Operations Centre established and staffed, infrastructure specified, purchased and installed, and finally middleware and management tools deployed. We consider that the use of these principles has been highly beneficial.

The Operations Centre hosts approximately 20 national servers, a certification TestGrid of approximately 40 machines and 4TB disk farm, a cluster of 64-CPUs and 4TB disk farm for the various testbeds, plus support staff. The TestGrid serves multiple purposes, but most importantly in the context of this paper, it acts as a non-reference porting platform.

3 Porting

3.1 Best Practice for Code Portability

Difficult issues arise when more than one programming language is allowed in a large project. At the other extreme, restriction to only one programming language is very difficult to enforce. The sheer size of the LCG2 and EGEE projects is such that no one person can know what everyone else is doing. Nonetheless, even without knowing a great deal about the software and its dependencies, one can formulate clear guidelines as to how to improve the software portability.

What follows is a summary based on the porting of software to non-reference platforms and what is described in the literature by experts in porting:

1. Dependency information: In large projects such as the LCG2/EGEE, new and legacy code should be designed in such a way that it is easily built by a third party. This requires that all dependencies and versions are described in some machine readable form. This was not, for example, the case for all packages in LCG2. Some dependencies were only required when building the package, but not required after the package was built.

2. Standards conformance: POSIX conformance [11] requires that POSIX standard code should be easily portable to other UNIX platforms. The use of non-POSIX standard code has always been problematic in our experience. ANSI and IEEE standards should always be adhered to.
3. Timing: Tsoukiàs[12] states very clearly and with good reason that lifetime, usability and reliability must be considered when developing and testing code. To develop software on too many platforms from the beginning is too costly and results in slow development. Some code may be a proof of concept, subsequently replaced or phased out. Expending too large an effort to get everything right first time often yields poor returns.
4. Build Tool portability: The build tools described by some authors are cumbersome but do have the ability to be extremely portable across all UNIX platforms. Ant is obviously portable also because it builds on Java. This is all good news.

3.2 Porting Objectives

The LCG2/EGEE software components are shown in the form of a dependency graph in Figure 1.

Grid-Ireland wished, in the first instance, that the porting of the LCG2 software to other platforms would focus on the ability to execute Globus and EDG jobs on worker nodes, and that replica management, R-GMA and VOMS would be supported.

To avail of the base functionality requires Globus and various EDG support packages. Since Globus 2.4.3 is known to have many bugs, the University of Wisconsin-Madison corrects these and packages all the necessary components as part of the Virtual Data Toolkit (VDT) [13]. We have assisted Maarten Litmaath in CERN to port VDT-1.1.14 to IRIX and Fedora Core 2. A Red Hat 9 port is already provided by VDT [14]. A port exists for Globus to Mac OS X and AIX but the VDT version must be ported to both of these platforms.

Grid-Ireland also wished that MPI, replica management and the OpenPBS client be provided on each worker node. In some cases Torque might be re-

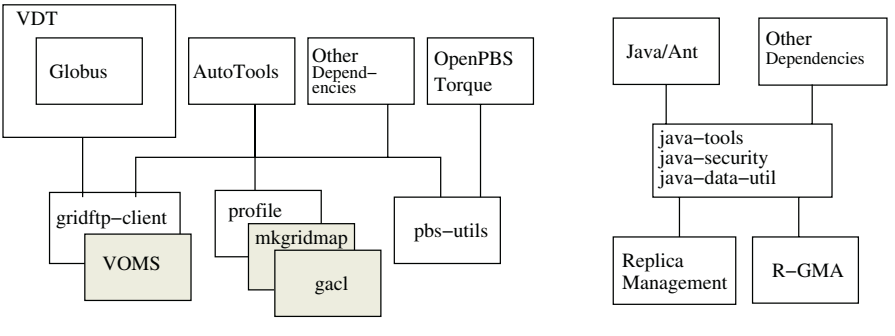


Fig. 1. LCG2/EGEE software components

quired since newer versions of operating systems are not always provided for in OpenPBS. Also the R-GMA information system producer and consumer APIs and the VOMS client were required.

At the moment there is no requirement by Grid-Ireland for the workload management system (WMS) but it appears that there is logging of WMS events from worker nodes to the resource broker. This logging activity can be disabled but with the loss of a desirable feature. WMS consists of many modules, but it may be able to be refactored to delineate those specific to the event logging, so that just this functionality needs to be ported. It should be noted that if the whole of WMS were ported successfully then almost everything will have been ported because it depends on so many other packages. Without WMS it will be shown that it is still possible to run jobs on the worker node (see Sect. 4.3).

There are a number of on-going issues, but we have successfully ported the functionality for job submission to Fedora Core 2, IRIX 6.5.14 and 6.5.17m, AIX 5.2L and Red Hat 9. We also plan to do this for Mac OS X v10.3 very soon, and a number of other platforms if the need arises within Grid-Ireland.

3.3 Porting of EGEE/LCG2

There were a large number of porting issues, some of which were trivial to solve, but others are on-going. The main issues were:

1. **Versioning issues:** The LCG2 C and C++ modules use Red Hat 7.3 autotools so that when porting to other platforms the exact same versions must be used so that the porting can be done successfully. For instance trying to use libtool-1.5.6 makes it impossible to compile VOMS under IRIX.
2. **Old package conflicts:** Most Linux operating systems are now updated using yum or apt and in the case of newer architectures some of the old Linux modules are considered obsolete. In the case of Fedora Core 2 the Red Hat 7.3 versions of automake and autoconf are contained in most yum repositories and happily co-exist with newer version.
3. **Binary Packaging issues:** Red Hat Package Manager (RPM) packages exist for most of the necessary software on Linux platforms. Packages have to be found in other formats for IRIX and AIX, which can be time consuming.
4. **Architecture Specific:** Some dependency packages are not available for certain architectures. For instance glibc has never been ported for later versions of gcc under AIX. Also gcc-3.2.2 is not available under AIX so gcc-3.3.3 has to be used instead. In the case of gsoap, the Linux version needs to be installed as gsoap-linux-2.3 instead of the less architecture specific gsoap-2.3.
5. **Shared libraries:** It was found that, under AIX, shared libraries could not be generated easily for some packages. Therefore the LCG2 code had to be altered to include static libraries instead.
6. **Differing bit architectures:** 32-bit terms are hard-coded in many packages, making the porting of software much more time consuming. Sufficient allowance is not always made for 64-bit and 32-bit variants.

7. **Dependency knowledge:** In many cases just trying to obtain the correct dependency packages and installing them in the correct locations was a very time consuming exercise.
8. **Module specific issues:** A brief summary of these issues are described as follows:
   (a) Only certain versions of the grid packaging toolkit (GPT) built on all platforms.
   (b) There was parsing issues when Globus was built under IRIX and AIX stopping the compilation from starting.
   (c) VDT required gnu *tar* since *tar* could not extract the package on all platforms correctly. Extra scripts were required under IRIX and Fedora Core 2 to create tar balls and RPMs.
   (d) gacl and edg-gridftp-client issues relate to shared libraries under IRIX.
   (e) VOMS required changes related to POSIX, compiler version, environment variable and 64-bit issues.
   (f) The order in which edg-build builds modules needed to be revised.
   (g) Finding the correct versions of software for replica management under IRIX took a considerable amount of time. Generally speaking, finding the correct dependencies for each module was very time consuming.

4 Autobuilding the Non-reference Ports

Early in the porting effort it became obvious that it would be necessary to make an exact replica of the DataGrid Red Hat 7.3 software repository so that any teething problems could be solved incrementally as the code was ported to other platforms.

At first the whole European DataGrid repository was obtained from the developer of the build software, Yannick Patois[15][16], of IN2P3. This software was used to maintain a Red Hat 6.2 and 7.3 repository of all the EDG software up to November 2003. The build software is now used to maintain the CrossGrid repository at FZK and the LCG2/EGEE repositories at CERN. CERN is using the auto-building tool to support Scientific Linux 3 (SL3) on both 32-bit and 64-bit architectures, as well as the Red Hat 7.3 version. Since April 2004 we are in constant contact with CERN with regard to porting the worker-node software to five other platforms. Currently we maintain a full copy of the EDG, LCG2 and some CrossGrid modules on a Red Hat 7.3 repository.

4.1 CVS Repositories

A number of CVS repositories are used to build all the necessary software for a worker node. The head version of VOMS is obtained from INFN's own repository. The whole of LCG2 is extracted using CVS checkouts directly from CERN's lcgware repository. The CrossGrid software is obtained by directly copying the CVS repository to a local repository. Edg-build then contacts this local repository to perform the nightly builds. The RAL repository of R-GMA will also

need to be added soon, since LCG2 no longer maintain the most recent version of R-GMA.

Because different repositories are being used to build different pieces of software, this means that there must be a number of different build machines, one for each repository. We have designed additional configuration scripts within edg-build for each type of CVS repository and also different start-up scripts for the cron jobs for each repository type. This makes the build process very easy to maintain and easy to port to other machines.

4.2 Auto-Build Porting Issues and Upgrades

The auto-build software, edg-build, has the ability to configure, compile and install software developed with the gnu autotools and apache ant. Therefore C, C++ and Java code can be compiled on a nightly basis using cron jobs or on a demand basis using a build-on-demand (BoD) server. The edg-build software is written in Python 2.2 making it portable to most operating systems. It contains facilities to run aclocal, automake, autoconf, libtool, make and ant in a sequential way, that will result in the production of binary tarballs or RPMs (under Linux). It was designed specifically to cater for gcc-2.95.2 and later gcc-3.2.2.

Although reasonably portable, edg-build had a number of issues and upgrades that needed to be taken care of before auto-building on non-reference platforms. These can be summarized as follows:

1. The software relies on specific versions of the gnu *autotools*. Any deviation from the specified list results in uncompilable code.
2. The Makefile could not be generated under IRIX or AIX, but it could be copied from Red Hat 7.3 and built the software successfully.
3. The Python scripts needed to be modified to allow for specific configurations of some modules.
4. Much of the LCG2/EGEE code relies on Red Hat 7.3 RPMs. Under Fedora these older RPMs are now deprecated. The older versions needed to be installed in a separate directory structure to allow edg-build to build the software correctly.
5. Direct access to most of the European CVS software repositories was impossible from inside our firewall, so a SSH tunnel was created to obtain access.
6. The sequence describing the compilation of nightly built modules was reordered so that every package built first time.

4.3 Results

In Figure 2 the current status of the build system can be seen. This figure is a snapshot of the Grid-Ireland auto-build web page [17] at the start of November 2004. The results change quite regularly as new ports are completed.

OS Type	Version	VDT	Basic	VOMS	RGMA	RM
Redhat	7.3	RPMS	RPMS	RPMS	RPMS	RPMS
Redhat	9.0	RPMS	RPMS	RPMS	RPMS	RPMS
Fedora Core	2	RPMS	RPMS	RPMS	tarball	tarball
					RPMS	RPMS
SGI	6.5.14	tarball	tarball	tarball	tarball	tarball
AIX	5.2L	tarball	tarball	tarball	tarball	tarball
Darwin	10	tarball	tarball	tarball	tarball	tarball

Colour	Meaning
	To be started
	Started
	Done

Fig. 2. Auto-build Results for Worker Nodes

5 Worker Node Setup

Currently our download repository contains only the Fedora Core 2 worker node RPMs. They can be deployed using yum, but we have found yum to be very slow in finding package dependencies, so we plan to migrate to apt. An empty RPM is provided which depends on the all the others, so performing an install of this one package will install the whole worker node. Post installation is achieved using the guidelines for LCG2 worker node installation[18]. At the moment this must be done by hand, but we plan to automate this step. In the case of IRIX where only tar balls are available, copying the */opt/* directory from one IRIX machine to another may be sufficient to create a working worker node, but this needs to be tested, particularly to see if libraries are missing or stored in other directories. One way to avoid this might be to create *tardist* files of all packages and ensure that all dependencies are accounted for on the new worker node.

6 Early Benchmarking Results

For comparison purposes a FFT job was submitted via Globus and via the EDG broker to worker nodes built with Fedora Core 2, IRIX, Red Hat 7.3 and Red Hat 9. These jobs involve submission of the executable built on the relevant worker node platforms. The executable is placed inside a wrapper script that is then described using the job description language (JDL). Mean times ($\bar{x}$) are

Table 1. Preliminary FFT EDG Job Submission Results

OS Type	Version	CPU Speed	iterations	$\bar{x}$	σ
Red Hat	9	2.8GHz	400	8.65	0.36
Red Hat	7.3	2.8GHz	400	8.39	0.53
Fedora Core	2	2.8GHz	400	9.6	0.34
IRIX	6.5.14	400MHz	100	10.52	0.01

calculated for a FFT with vector length of $N = 2^{16}$ for a specified number of iteration (see Table 1). Red Hat 7.3 turns out to be the faster than Red Hat 9, and Red Hat 9 faster than Fedora Core 2. IRIX is approximately 5 times slower but has a much slower CPU speed. The load is very stable on the IRIX as can be seen by the small standard deviation (σ). The mean and standard deviation are calculated from 10 samples.

7 Conclusions

The base worker node port of the LCG2/EGEE grid software for Globus and EDG job submission is now completed for Fedora Core 2, Red Hat 9, IRIX and AIX. VOMS is ported to Fedora Core 2, Red Hat 9 and IRIX but still has pending issues related to 64-bit architectures, amongst other things. In the coming months VOMS will be supported on the LCG2/EGEE testbed, so this issue, we presume, will be resolved as time goes on. We also have R-GMA and replica management software building under Fedora Core 2 and Red Hat 9 but it will take a little longer to complete under IRIX and AIX. The workload management system will possibly be incorporated, or some subset of it, in the foreseeable future. We plan to release software tags that are exactly the same versions as those provided for SL3, and do this using apt repositories. This requires that the replica management be tested for all worker nodes in accordance with the test procedures for deployment laid out in the LCG2 deployment documentation [19].

This has allowed us to begin a most interesting set of benchmarking and heterogeneity experiments involving all of these platforms. With manual installation of Fedora Core 2, IRIX and Red Hat 9 worker nodes it will be possible to submit jobs heterogeneously across Grid-Ireland.

Acknowledgements

We would like to thank IBM and Dell for sponsoring us with machines to perform the software ports, and Science Foundation Ireland for funding this effort. We would also like to thank Vincenzo Ciaschini in INFN for his sterling work on VOMS. We gratefully thank DIAS for the SGI machine they have loaned to us for the IRIX port. Most of all we would like to thank the deployment group in CERN for all their help in porting to each platform, in particular Carlos Osuna for his help in customizing the auto-build procedure and Maarten Litmaath in porting VDT Globus.

References

1. LHC: Large hadron collider computing grid project. http://lcg.web.cern.ch/LCG/ (2004)
2. EDG: European datagrid project. http://www.eu-datagrid.org/ (2004)

3. EGEE: Enabling grids for e-science in europe. http://www.eu-egee.org/ (2004)
4. CrossGrid. http://www.crossgrid.org/ (2004)
5. CASEY, J., BAUD, J.P.: The Evolution of Data Management in LCG-2. http://chep2004.web.cern.ch/chep2004/ (2004)
6. Maroney, O.: London Tier-2. http://www.gridpp.ac.uk/gridpp11/ (2004)
7. Coghlan, B., Walsh, J., Quigley, G., O'Callaghan, D., Childs, S., Kenny, E.: Principles of Transactional Grid Deployment. Submitted to EGC 2005 (2004)
8. Lastovetsky, A.: Adaptive parallel computing on heterogeneous networks with mpC. Volume 28. Elsevier Science Publishers B. V. (2002)
9. Ryan, J.: SMG: Shared Memory for Grids. PDCS'04, Boston (2004)
10. Morrison, J.: Condensed Graphs: Unifying Availability-Driven, Coercion-Driven, and Control-Driven Computing. ISBN: 90-386-0478-5 (1996)
11. Lynuxworks: POSIX conformance. http://www.lynuxworks.com/products/posix/posix.pdf (2004)
12. Stamelos, I., Vlahavas, I., Refanidis, I., Tsoukiàs, A.: Knowledge Based Evaluation of Software Systems: a Case Study. http://l1.lamsade.dauphine.fr/tsoukias/papers/kb-esse.pdf (2003)
13. VDT: Virtual data toolkit. http://www.cs.wisc.edu/vdt/ (2004)
14. VDT: Versions for RH7.3 and RH9.0. http://www.cs.wisc.edu/vdt/vdt_rpms/1.2.1/stable/ (2004)
15. Garcia, A., Hardt, M., Patois, Y., Schwickerath, U.: Collaborative Development Tools. Cracow Grid Workshop (2003)
16. Patois, Y.: Datagrid configuration management and build conventions. http://datagrid.in2p3.fr/d6.2/DataGrid-ConfMngmt-BuildConv.html (2003)
17. Autobuild: TCD Autobuild Web-page. http://grid.ie/autobuild (2004)
18. Retico, A., Usai, A., Keeble, O., Diez-Andino, G.: LCG Grid INfrastructure Support. http://grid-deployment.web.cern.ch/grid-deployment/gis/lcg-2_2_0/LCG2InstallNotes.pdf (2004)
19. LCG2-deployment: Deployment web-pages. http://cern.ch/grid-deployment/ (2004)

Effective Job Management in the Virtual Laboratory

Marcin Lawenda, Norbert Meyer, Maciej Stroiński, Tomasz Rajtar, Marcin Okoń, Dominik Stokłosa, and Damian Kaliszan

Poznań Supercomputing and Networking Center,
Noskowskiego 10, 61-704 Poznań, Poland
Marcin.Lawenda@man.poznan.pl
http://vlab.psnc.pl

Abstract. In the paper approach to effective job management in the virtual laboratory is presented. The Dynamic Measurement Scenario (DMS) is used for advanced workflow control. DMS often consists of many different tasks connected by links with specified conditions. To prepare DMS we need a special application (during the definition process) on the one hand and on the other hand a sophisticated (and complex) module for manage and control specified workflow is needed. The described issues are illustrated by Nuclear Magnetic Resonance (NMR) laboratory examples.

1 Introduction

Experiments executed in the science laboratories are complex and consist of many stages [3],[4]. Usually it looks as follows: a scientist prepares a sample and/or input data (e.g. parameters) which will be measured/computed. Next she/he uses laboratory devices to achieve data which are the processed by a specialized software. Processing can include the visualization stage if it is needed to assess the results. In case of undesirable results some measurement stages should be repeated. At each stage the scientist decides which way the research should go next. As we can see, the experiment process execution may consist of very similar stages in many scientific disciplines (laboratories). The given experimental process is often executed recurrently by some parameters modification. Obviously, the presented scenario is typical but we realize that scholars can have more sophisticated needs.

Thus, we can define a graph which describes the execution path specified by a user. Nodes in this graph correspond to experimental or computational tasks. Edges (links) correspond to the path the measurement execution is following. In nodes we have a defined type of application and its parameters. In links the passing conditions which substitute decisions made by the user are defined. These conditions are connected with applications and let us determine if the application is finished with the desired results (if not, the condition is not met). This execution graph with specified nodes and links (with conditions) is

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 414–423, 2005.

called the Dynamic Measurement Scenario (DMS). The term "dynamic" is used, because the real execution path is determined by the results achieved on each measurement stage and can change *dynamically*. All the DMS parameters can be written down using a special language: the Dynamic Measurement Scenario Language (DMSL).

The DMS is used in the Virtual Laboratory [5] system developed in Poznań Supercomputing and Networking Center in collaboration with the Institute of Bioorganic Chemistry (IBCH) and Department of Radio Astronomy of Nicolaus Copernicus University.

2 Related Work

Pegasus. Pegasus (Planning for Execution in Grids) [1], was developed at Information Science Institute (ISI) as part of the GriPhyN project [2]. The system is designed to map abstract workflows onto the Grid environment. The abstract workflow is presented as an workflow, where the activities are independent of the Grid resources used to execute those activities. The abstract workflow can be constructed by Chimera or can be written directly by the user. Whether the input comes through Chimera or is given directly by the user, Pegasus requires that it is in DAX format. Based on this specification, Pegasus produces a concrete (executable) workflow.

Wf-XML. Wf-XML comes from the Workflow Management Coalition (WMC), an independent body of workflow vendors, customers and higher education establishments. The Wf-XML interface is based on XML and build on top of the Asynchronous Service Access Protocol (ASAP), which is in turn built on Simple Object Access Protocol (SOAP). It allows workflows from different vendors to communicate with each other. The WMC claims that Wf-Xml is an ideal way for the Business Process Manager (BMP) engine to invoke a process in another BPM engine. Such engines have the property that their process can be examined by being retrieved in a standard form, such BPEL. It also provides an interface to send new or updated process definitions to the BPM engine.

The VLab has different approach. First of all, we do not want to target only the computational applications, but also live experiments and interactive and batch applications. The VLab user is given a simple graphical interface (Scenario Submission Application - SSA) and is not required to study new language. We have created a resource schema which can be used to describe resources like experiments (i.e. NMR spectroscopy, radio telescope, etc.) or computations. Using SSA the VLab user builds his workflow, which is called Dynamic Measurement Scenario (DMS), connecting defined resources together and providing the required properties. Moreover each application or experiment in DMS is represented as an separate node. The DMS can be submitted to the VLab system, where is decomposed according to the user specification and each node is launched. When

the current node is experiment the user is given an interface to the scientific equipment where he can conduct an experiment. The computational nodes are submitted to the GRID using GRMS Broker [6]. The decision, where the task will be executed is made basing on the user specification.

3 Designing

To properly define the Dynamic Measurement Scenario the system has to have knowledge about available applications and connections which are enabled to create. In case of the laboratory type, where the processing software is well known, it seems to be easy. The issue will be more complex when we want to define the DMS model for a wider range of virtual laboratories.

To solve this problem we have defined a special language (DMSL) which determines the following conditions: names of the connected applications, a condition (or conditions) connected with a given path, an additional condition which has to be met to pass a given path (e.g. when special conversion between application is needed) and, finally, a list of input or output files for applications.

An expertise from a particular discipline is necessary to define rules for DMS. It can be done by a laboratory administrator together with a domain expert.

The DMS schema must be designed by the VLab administrator before it is available for users. Creating a new Dynamic Measurement Scenario needs a special attitude. We assume that some stages will be performed by the computer science engineer and scientist from a scientific discipline DMS concerns, and some by application.

The designing of the DMS consists of the following stages:

- application analyzing,
- connection diagram preparing,
- describing additional dependencies in the connection diagram,
- applications and links description generating,
- measurement scenario description generating.

All stages are tightly connected between themselves and should be executed in the presented sequence. The first three phases should be performed, as we mentioned before, by the VLab administrator and engaged scientist (see 3.1, 3.2, 3.3). The last two can be done by a special application which take into consideration user's rights and information introduced by the user (see 3.4, 3.5). Next, we will analyse each stage of DMS designing. Because of the paper limitations we had to focus only on the major aspects of these issues.

3.1 Application Analyzing

Each application available in the dynamic scenario must be first analyzed from the functional point of view. Input and output parameters have to be taken into consideration. Also, input and output format files must be described. An exemplary option to describe in these files follows: file type (binary, text), filename

format (if exists): filename mask, filename extension. If the file format type is text, the analyst can analyze information within. A specified pattern connected with parameters important for the final user (calculated by application) can be used to specify link conditions.

3.2 Connection Diagram Preparation

After the application analysis step the VLab administrator can make the connection diagram. In this phase applications are assigned to execution stages. Exemplary stages for the laboratory of NMR spectroscopy follow: acquisition, processing, visualization, structure determination, structure visualization.

The following stages do not have to occur one after another in the measurement process but connection dependencies have to be met. The available measurement sequence is determined in the next step. Each application from each stage is connected to another one taking into consideration data achieved in the previous analysing step (see 3.1).

At this stage the necessity of cooperation between a computer science engineer and a scientist connected with a given science discipline is strictly required.

In Fig. 1 we present an exemplary diagram for the Virtual Laboratory of NMR Spectroscopy.

3.3 Describing Additional Dependencies in the Connection Diagram

At this stage we present conditions on the connections between applications which were created at the previous stage (see 3.1). We should focus our attention on: connection conditions, conversion issues, files types related to links.

Conditions defined on the connections can influence the measurement execution path. After the end of each application execution they are verified and in this way the following execution path is determined. For more information please go to 4.

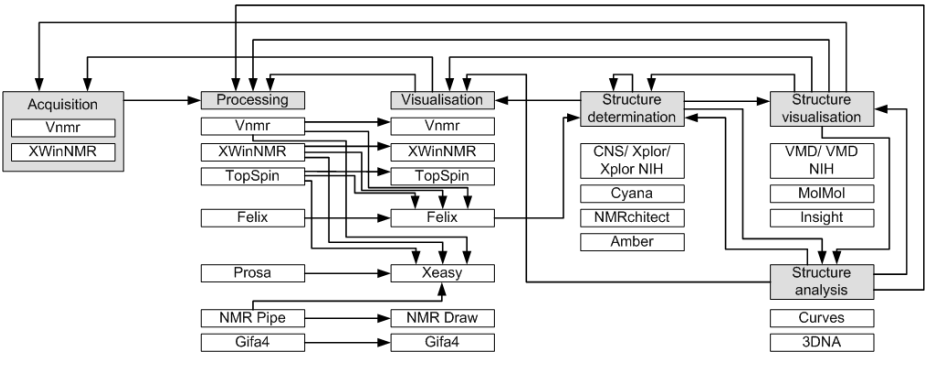


Fig. 1. An exemplary connection diagram for the Virtual Laboratory of NMR Spectroscopy (prepared by Łukasz Popenda (IBCH))

Conversion is performed when two connected applications have a different input-output file format. It should be described for each connection separately. Depending on the connected applications conversion can be realized in a different way. More about the conversion issues can be found in section 5.

It is necessary to determine which type of file can be used as an input file to the target application. Therefore, a set of output files of a given application must be related to each link. Note that a different set of files can be related to each link. This means that some output files of a source application can be used for one target application but not for another one and vise versa.

3.4 Applications and Links Description Generation

The next steps can be performed using the Scenario Submission Application (SSA) based on the information from the previous stages. DMS is encoded in the Dynamic Measurement Scenario Language (DMSL). DMSL base on the XML and XSD standard. The general DMS consists of a description of all possible applications with all parameters available for users.

XSD Schema. The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD. An XML Schema defines: elements that can appear in a document, attributes that can appear in a document, defines which elements are child elements, defines the order of child elements, defines the number of child elements, defines whether an element is empty or can include text, defines data types for elements and attributes, defines default and fixed values for elements and attributes. The XML Schema language is also referred to as the XML Schema Definition (XSD). The next paragraphs describe the XSD schemas which are used by the VLab system. The XSD schemas have been created for the Dynamic Measurement Scenario definition.

Components Description Schema. The Components Description Schema (CDS) defines the Java swing components (JComponent) which are used for displaying the resource properties (See RDS description). The schema structure is presented in a diagram in Fig. 2.

Also, a set of component attributes is available. Some of the attributes are required by the component, some are optional. They are all used to define the component behaviour i.e. whether the component should be visible, editable, etc. A list of all available attributes is presented in the table below.

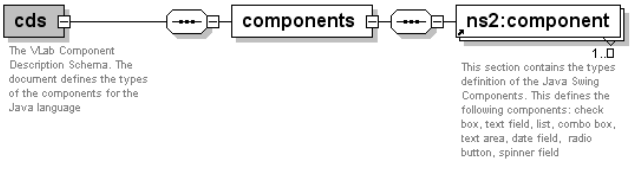


Fig. 2. Component description schema

No	Name	Description
1	id	the component identifier
2	name	component name
3	class	java class name which implements the given component
4	modelDataAttached	specifies whether the component requires the model data (i.e. JList requires the list of items. From the set of items the user chooses the item)
5	model	the java class which implement the component data model
6	document	the java class, which implements the component document
7	editor	the java class, which implements the component editor
8	enabled	determines whether this component is enabled. An enabled component can respond to user input and generate events
9	visible	Determines whether this component should be visible when its parent is visible
10	editable	tells whether the component is read only
11	columns	the number of columns (i.e. in the JTextarea)
12	rows	the number of rows (i.e. in the JTextarea)
13	min	the minimum value of the component (i.e. JSpinner)
14	max	the maximum value of the component (i.e. JSpinner)
15	step	the step value of the component (i.e. JSpinner)
16	selectedIndex	the index of the component item which should be selected by default (i.e. JComboBox)
17	dateFormat	specifies the date format of the component
18	icon	the component icon (i.e. JCheckBox)
19	selectedIcon	the component selected icon (i.e. JCheckBox)

Using the schema described above the following Java components have been defined: check box (JCheckBox), text field (JTextField), list (JList), combo box (JComboBox), text area (JTextArea), date field, radio button (JRadioButton), spinner field (JSpinner).

Links Description Schema (LDS). The LDS schema describes the available connections between resources. It also specifies the conditions which have to be taken into consideration while connecting resources. The schema structure is presented below (Fig. 3).

The Link Element. The link element, as it is shown in the figure below contains the connection definition between two nodes (Fig. 4).

The connection attributes are: id (the link identifier), resourceId (component name), externalConversion (necessity of external conversion).

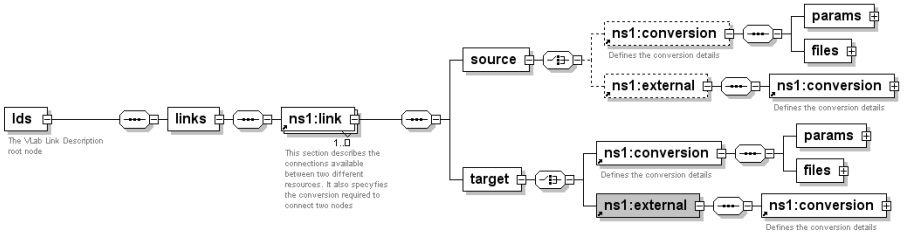


Fig. 3. Links description schema

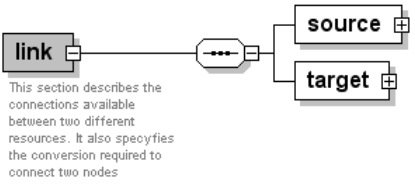


Fig. 4. The link element

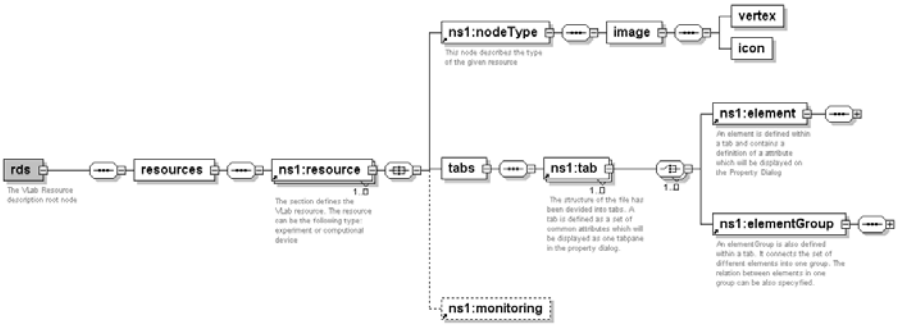


Fig. 5. Resource description schema

Resource Description Schema. The Resource Description Schema (RDS) has been created for the VLab resource description. The general resource description structure is presented in the figure below (Fig. 5).

The schema defines a list of the resources which are all attached to the resources node. Every resource element contains the following sections: nodeType (defines the type of the given resource), tabs (the resource properties are grouped in tabbed panes which are displayed in the JTabbedPanes), monitoring (information about the given resource in the VLab system).

The resource definition can be visualized by the Scenario Submission Application.

3.5 Measurement Scenario Description

The user is welcome to create the measurement diagram using the Scenario Submission Application (SSA).

The applications and links definition specified in the previous stage are imported to the SSA. A List of imported applications take into account the user's rights. Thus, it consists of only these applications descriptions (and possible connections) which are available for the user.

After finishing the measurement scenario defined by the user, a new DMS description is generated. It is defined on the basis of the diagram and information added by user. Only the chosen application descriptions with specified parameters are placed there.

4 Connection Issues

The user can define conditions on the connections which can influence the measurement workflow. These conditions concern the computational or experimental results. A set of available conditions to be specified by the user (while a new measurement is defined) is determined on the "describing additional dependencies" stage of the DMS creation process (see 3.3). To enable a given connection all conditions defined there have to be met.

The user is also welcome to define logical conditions between links. There are two types of conditions: *OR* (default) and *AND*. Logical conditions can be defined in the beginning and end of connections. Their meaning depends on their localization.

Beginning Conditions. The Beginning conditions are connected with the results achieved at the computational stage. Each link is marked by logical ***1*** if all conditions are met or logical ***0*** otherwise. The default logical operation defined between links is *OR*, which means that each execution path can be done separately. In case of the *AND* operation simultaneously execution of two or more paths is possible on condition that achieving results in the source application are met (value ***1*** on each link).

Ending Conditions. The ending conditions are defined to assure that all necessary data is available before the start of the next processing task.

Similarly as before, for the *OR* condition the link is independent from the other ones. This means that if conditions on a given link are met, the measurement process can go on without waiting for other processes.

The *AND* condition is used when results computation from a few sources is needed. In this way we can force waiting for finishing a number of processes and only then run the next task.

The discussed situation is presented in Fig. 6.

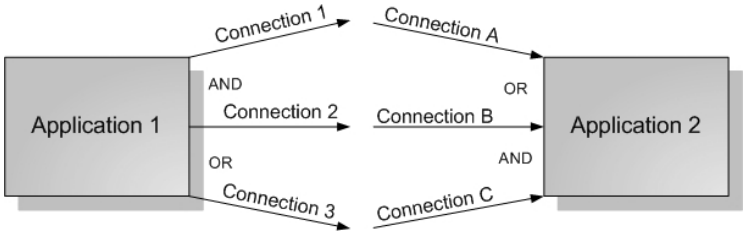


Fig. 6. Defining in/out-connections conditions

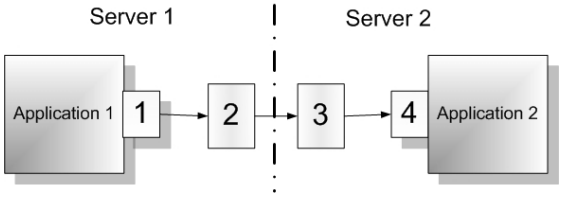


Fig. 7. Four ways of conversion

5 Conversion Issues

In this section we want to focus on the conversion issues. As we mentioned before, conversion is made when input and output file application formats are different. We assume that conversion may be done in four different ways:

- setting up the switch in the source application to export data in an appropriate format (1),
- using a program on the source server to convert the output file (mini post-processing) (2),
- using a program on the source server to convert the output file (mini pre-processing) (3),
- setting up the switch in the target application to import data in an appropriate format (4).

The discussed situation is illustrated on figure below (Fig. 7).

The conversion way has to be matched with a given connection. The preferred way is using the application switches because it usually does not cause additional CPU power consuming. Please note that not every conversion way will be possible in each situation.

6 Summary

The Dynamic Measurement Scenario is a flexible and very convenient way to define workflow in many types of virtual laboratories. It allows to spare a lot

of user's time and speed up of the project realization. It facilitates defining and monitoring the measurement process from the preparation stage through experimental and computational processes to results analysis (based on the achieved visualization data). In the DMS model, besides the definition of the tasks execution sequence, we can also define some extra connections (e.g. loops, parallel connections), conditions on the connections and different lengths of the execution paths. Thanks to the possibility of the conditions defining on the connections paths, the real path is determined during execution and can depend on computational results. Thanks to the Submission Scenario Application the user can easily define, submit and monitor the progress of the DMS realization.

References

1. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Mei-Hui Su, Vahi, K., Livny, M.,: Pegasus: Mapping Scientific Workflows onto the Grid, Across Grids Conference 2004, Nicosia, Cyprus
2. Deelman, E., Blythe, J., Gil, Y., Kesselman, C.,: Workflow Management in GriPhyN, Grid Resource Management, J. Nabrzyski, J. Schopf, and J. Weglarz editors, Kluwer, 2003.
3. Lawenda, M., Meyer, N., Rajtar, T.: General framework for Virtual Laboratory. The 2nd Cracow Grid Workshop, Cracow, Poland, December 11 - 14, 2002
4. Lawenda, M., Meyer, N., Rajtar, T., Okoń, M., Stokłosa, D., Stroiński, M., Popenda, Ł., Gdaniec, Z., Adamiak, R.W.: General Conception of the Virtual Laboratory. International Conference on Computational Science 2004, LNCS 3038, pp. 1013-1016, Cracow, Poland, June 6-9, 2004
5. Virtual Laboratory project `http://vlab.psnc.pl/`
6. WP9 Resource Management - GridLab Project `http://www.gridlab.org/WorkPackages/wp-9/`

Workflow Management in the CrossGrid Project*

Anna Morajko[1], Enol Fernández[1], Alvaro Fernández[2],
Elisa Heymann[1], and Miquel Ángel Senar[1]

[1] Universitat Autònoma de Barcelona, Barcelona, Spain
{ania, enol}@aomail.uab.es
{elisa.heymann, miquelangel.senar}@uab.es
[2] Instituto de Física Corpuscular, Valencia, Spain
alvaro.fernandez@ific.uv.es

Abstract. Grid systems offer high computing capabilities that are used in many scientific research fields and thus many applications are submitted to these powerful systems. Parallel applications and applications consisting of inter-dependent jobs may especially be characterized by a complex workflow. Therefore, Grid systems should be capable of executing and controlling workflow computations. This document sets out our approach to workflow management in a Grid environment. It introduces common steps on how to map an application workflow to the DAG structure, and how to carry out its execution and control. We present the Workflow Management Service (WFMS) implemented and integrated as a part of the CrossGrid project. The purpose of this service is to schedule workflow computations according to user-defined requirements, also providing a set of mechanisms to deal with failures in Grid.

1 Introduction

The Grid represents distributed and heterogeneous systems and involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations [1]. Grid systems offer high computing capabilities that are used in many scientific research fields. They facilitate the determination of the human genome, computing atomic interactions or simulating the evolution of the universe. Many researchers have therefore become intensive users of applications with high performance computing characteristics. There are projects such as GriPhyn [2], DataGrid [3], GridLab [4] or Crossgrid [5] that provide the middleware infrastructure to simplify application deployment on computational grids.

The main objective of the CrossGrid project is to incorporate a collection of machines distributed across Europe, and to provide support especially for parallel and interactive compute- and data-intensive applications. As a result of this project, parallel applications compiled with the MPICH library (and using either ch-p4 [6] or Globus2 [7] device) are executed on Grid resources in a transparent and automatic way. The workload management system that we have implemented as part of the

* This work has been partially supported by the European Union through the IST-2001-32243 project "CrossGrid" and partially supported by the *Comisión Interministerial de Ciencia y Tecnología* (CICYT) under contract TIC2001-2592.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 424 – 433, 2005.

CrossGrid middleware carries out all necessary steps incurred from the time that the application is submitted by the user until the end of its execution (i.e. potential resource discovery, selection of the best matched resources and execution tracking). Our workload management system has been designed to manage Grid-specific details of the application execution with minimal effort by the user.

In the previous work [8], we described the specific details related to the execution of MPI applications on the Grid. In this paper, we focus on the additional service that we have included to support workflow computations. Many applications may consist of inter-dependent jobs, where information or tasks are passed from one job to another for action, according to a set of rules. Such applications known as workflows consists of a collection of jobs that need to be executed in a partial order determined by control and data dependencies. Workflows are an important class of applications that can take advantages of the resource power available in Grid infrastructures, as has been shown in the LIGO [9] pulsar search, several image processing applications [10] or physics experiment ATLAS [11]. The execution of such an application may be very difficult. Normally, a user should submit to a Grid system manually, job by job, following the rules of dependencies that appear between jobs. The manual tracking of the application workflow may be very ineffective, time consuming and may produce many errors in application execution. Therefore, we present a solution to the automatic management of the application workflows applied in the CrossGrid project.

Section 2 briefly presents related work. Section 3 introduces a general overview of the workload management in the CrossGrid, indicating new features for workflow support. Section 4 sets out the workflow notation and Section 5 introduces details of the workflow management. Section 6 shows the results of the probes conducted using a workflow whose structure is representative of the ATLAS experiment. Finally, Section 7 presents the conclusions to this study.

2 Related Work

A number of studies in Grid systems provide general-purpose workflow management. The Condor's DAGMan [12] – DAGMan (Directed Acyclic Graph Manager) is a meta-scheduler for Condor. DAGMan manages dependencies between jobs, submitting these to Condor according to the order represented by a DAG and processes the results. The DAG must be described in an input file processed by the DAGMan and each node (program) in the DAG needs its own Condor submit description file. DAGMan is responsible for scheduling, recovery, and reporting for the set of programs submitted to Condor. This scheduler focuses on the execution of workflows in a local cluster managed by the Condor system.

Pegasus system [13] – Planning for Execution in Grids – was developed as part of the GriPhyN project. Pegasus can map scientific workflows onto the Grid. It has been integrated with the GriPhyN Chimera system. Chimera generates an abstract workflow (AW), Pegasus then receives such a description and produces a concrete workflow (CW), which specifies the location of the data and the execution platforms. Finally, Pegasus submits CW to Condor's DAGMan for execution. This system focuses on the concept of virtual data and workflow reduction. Triana [14] is a Problem Solving Environment (PSE) that provides a graphical user interface to

compose scientific applications. A component in Triana is the smallest unit of execution written as Java class. Each component has a definition encoded in XML. Such created application's graph can then be executed over Grid network using the GAP interface. Unicore [15] stands for Uniform Interface to Computing Resources and allows users to create and manage batch jobs that can be executed on different systems and different UNICORE sites. The user creates an abstract representation of the job group (AJO – Abstract Job Object) that is then serialized as a Java object, and in XML format. UNICORE supports dependencies inside the job group and ensures the correct order of the execution. Its job model can be described as directed acyclic graphs. UNICORE maps the user request to system specification, providing job control. In contrast to our work, Triana, Pegasus and Unicore lack resource brokerage and scheduling strategies.

GridFlow [16] supports a workflow management system for grid computing. It includes a user portal in addition to services of global grid workflow management and local grid sub-workflow scheduling. At the global level, the GridFlow project provides execution and monitoring functionalities. It also manages the workflow simulation that takes place before the workflow is actually executed. This approach is applicable only in the case of having performance information about job execution. At the local grid sub-workflow level, scheduling service is supported.

3 CrossGrid Workload Management

This section presents the main components that constitute the Workload Management System (WMS) applied in the CrossGrid project. A user submits a job to a Scheduling Agent (SA) through a Migrating Desktop or command line (see Figure 1). The job is described by a JobAd (Job Advertisement) using the EU-Datagrid Job Description Language (JDL) [17], which has been extended with additional attributes to support interactive and parallel applications, as well as workflows.

To support the workflow execution, we have included specific service into the WMS. Workflows have a special treatment at the beginning as they are passed from the SA to the Condor's DAGMan, which is a specialized scheduler module that submits each individual job to the SA when job dependencies have been satisfied.

For each simple job (submitted directly by the user or by the Condor's DAGMan), the SA follows the same steps. It asks the Resource Searcher (RS) for resources to run the application. The main duty of the RS is to perform the matchmaking between job needs and available resources. The RS receives a job description as input, and, as output, returns a list of possible resources within which to execute the job. Computing resources are available as Computing Elements (CE), which provide the abstraction of a local farm of Working Nodes (WN). This local farm (or CE) is accessed through a Gatekeeper. The list of resources returned by the Resource Searcher consists of a Computing Elements list. Subsequently, the Scheduling Agent selects a CE on which to run the job. The SA passes the job and the first-selected CE to the Application Launcher (AL), who is responsible for the submission of that job on the specified CE. The AL passes the job to Condor_G [18], which manages a queue of jobs and resources from sites where the job can be executed. Due to the dynamic nature of the Grid, the job submission may fail on that particular CE. Therefore, the Scheduling

Agent will try the other CEs from the list returned by the Resource Searcher. Finally, the Scheduling Agent notifies the user of the result.

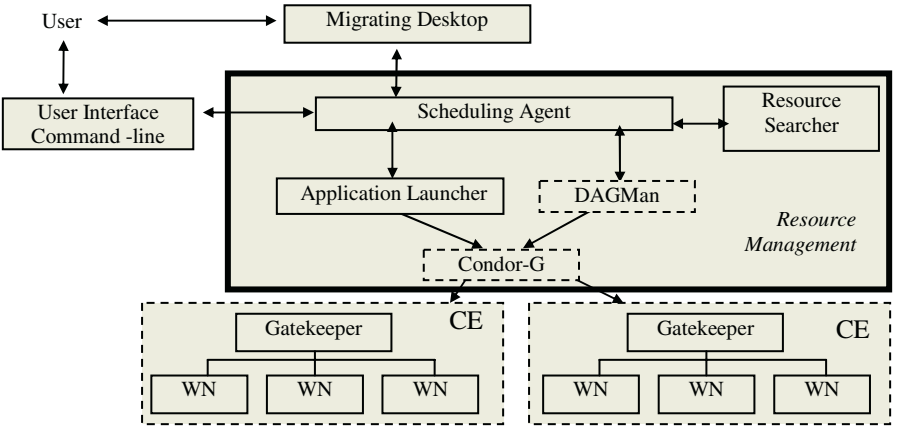


Fig. 1. Architecture of the Workload Management System

4 Workflow Notation and Specification

There are many complex applications that consist of inter-dependent jobs that cooperate to solve a particular problem. The completion of a particular job is the condition needed to start the execution of jobs that depend upon it. This kind of application workflow may be represented in the form of a DAG – a directed acyclic graph. A DAG is a graph with one-way edges that does not contain cycles. It can be used to represent a set of programs where the input, output, or execution of one or more programs is dependent on one or more other programs. The programs are nodes (vertices) in the graph, and the edges (arcs) identify the dependencies of these programs. Figure 2 presents an example DAG that consists of 4 nodes lying on 3 levels. The execution of the indicated DAG consists of three successive steps:

1. Execution of the node NodeA from the first level.
2. Parallel execution of nodes NodeB1 and NodeB2 from the second level. The execution can start if and only if the execution of the node NodeA is successful.
3. Execution of the node NodeC from the third level. The execution can start if and only if the execution of all nodes from the level two is successful.

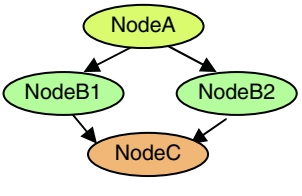


Fig. 2. Example DAG

4.1 Workflow Specification Using JDL

Workflows, similar to a normal job, are specified in a text file using DataGrid JDL format extended appropriately to support DAG structure and control. The workflow description has two components: specification of dependencies between computations (node dependencies) and specification of computation (node description). Below, we present an example JDL file for a workflow specified as in Figure 2.

```
[
type = "dag";
nodes = [
     dependencies={{NodeA,{NodeB1,NodeB2}},{{NodeB1,NodeB2},NodeC}};
     NodeA = [
            node_type = "edg-jdl";
            description = [
                   Executable = "jobA.sh";
                   InputSandbox = {" jobA.sh"};
            ];
     ];
     NodeB1 = [
            node_type = "edg-jdl";
            node_retry_count = 3;
            app_exit_code = { 10, 11 };
            file = "jobB1.jdl";
     ];
     NodeB2 = [
            node_type = "edg-jdl";
            file = "jobB2.jdl";
     ];
     NodeC = [
            node_type = "edg-jdl";
            file = "jobC.jdl";
     ];
];
]
```

Job specification

Node specification

4.2 Dependencies Between Nodes

Each dependence is specified as a pair of elements positioned between brackets, where the meaning is that the second element depends on the first. Both elements may be formed by a set of elements written between brackets. This indicates a dependence of many-to-one, one-to many or many-to-many elements. Therefore, considering the example DAG, there are few possibilities to specify the attribute *dependencies*:

- `{{NodeA,NodeB1},{NodeA,NodeB2},{NodeB1,NodeC},{NodeB2,NodeC}}`
- `{{NodeA,NodeB1},{NodeA,NodeB2},{{NodeB1,NodeB2},NodeC}}`
- `{{NodeA,{NodeB1,NodeB2}},{{NodeB1,NodeB2},NodeC}}`

4.3 Node Description

The attribute `nodes` contains the list of nodes that form the DAG. Each node represents a job to be executed and contains node-specific attributes, as well as the job specification. The attributes for a node description are:

- `node_type` – specifying a type of a node. This attribute is mandatory and must contain the value "edg-jdl", as the node is a normal job written in JDL format.
- `node_retry_count` – specifying how many times a node execution may be retried in the case of failure. This attribute is optional. If the user specifies this attribute for a node; hence, if it fails, this particular node will be automatically retried. Otherwise, this attribute will be set to the default value.
- `app_exit_code` – specifying the possible exit codes for a job. If a node fails because of the application failure (e.g. segmentation fault, division by 0, a file already registered in a Storage Element), then a job should be aborted. However, when a job fails because given resources fail (e.g. a machine failure, Condor/PBS queue problems), it should be automatically retried. By default, in both cases, the node will be retried until `node_retry_count`. The attribute `app_exit_code` provides the possibility to control the job exit code and terminate job execution in case of failure. If this attribute contains certain values, this means that the job may return such values and that they are recognized by the user. If the job execution returns one of the specified values, the node will not be retried. Otherwise, the job will be retried automatically according to `node_retry_count` (if the maximum of retries is not reached) and submitted to other CE.
- `description / file` – specification of the job; A job can be a normal, single job or an MPI job. There are two ways to specify a job:
  - Via an attribute called `description`, where a job is specified directly inside this attribute in JDL:

```
description = [
        Executable = "jobA.sh";
        InputSandbox = {" jobA.sh"};
        …
];
```

  - Via an attribute `file`, where a job is specified in the indicated JDL file

```
file = "jobA.jdl";
```

5 Scheduling Application Workflow

To support application workflows, we have extended the Scheduling Agent with a new component called Workflow Manager Service (WFMS). The WFMS is executed within the Resource Broker machine that contains the WMS presented in Section 3. Specification and implementation of the workflow supported in the CrossGrid project takes advantages and leverages of a preliminary work presented in [19]. As shown in Figure 1, workflows are passed from the Scheduler Agent to the Condor's DAGMan. DAGMan is an iterator on the DAG, whose main purpose is to navigate through the graph, determine which nodes are free of dependencies, and follow the execution of corresponding jobs, submitting these to the SA. While DAGMan provides us with the automatic graph management, the WFMS is responsible for searching grid resources, controlling errors and retrying the execution of failed nodes avoiding CEs' repetition. A DAGMan process is started for each workflow submitted to the WMS. If there is more than one DAG to execute, a separate DAGMan process will be started for each DAG. A set of steps must be performed for each node in the workflow:

1. Initial phase – preparing all necessary information for the node execution. A suitable resource to run the job is searched by the Resource Searcher and the job is then passed to Condor-G, which will be responsible for submitting this to the remote site. If no resources are found, the WFMS will mark the node as failed, which implies the end of the node execution. This node can be automatically retried according to the `node_retry_count` value.
2. Job execution on the remote site.
3. Final phase – checking the job execution return code. If the job was executed successfully, it is marked as Done. Otherwise, the WFMS compares the return value to the attribute `app_exit_code`; if the return value is one of the values specified by the user, the job is not retried; in any other, case the job is marked as failed and is retried according to the `node_retry_count` value.

The WMS also supports the functionality by which to submit a failed workflow and execute only those nodes that have not yet been successfully executed. This workflow is automatically produced by the WMS when one or more nodes in the workflow has resulted in failure, making the application execution impossible to finish. If any node in the a workflow fails, the remainder of the DAG is continued until no more forward progress can be made, due to workflow's dependencies. At this point, the WFMS produces a file called a Rescue DAG, which is given back to the user. Such a DAG is the same as the original workflow file, but is annotated with an indication of successfully completed nodes using the `status=done` attribute. If the Rescue DAG is resubmitted using this Rescue DAG input file, the nodes marked as completed will not be re-executed.

A DAG is considered as a normal, single job unit of work. A DAG execution goes through a number of states during its lifetime:

- *Submitted* – The user has submitted the DAG using User Interface but it has not yet been processed by the Network Server
- *Waiting* – The DAG has been accepted by the Network Server
- *Ready* – The DAG has been processed by the Workload Manager that has decided to run a Condor's DAGMan
- *Running* – The DAGMan process is running; the DAG is passed to the DAGMan
- *Done* – The DAG execution has finished
- *Aborted* – The DAG execution has been aborted because of an external reason
- *Cancelled* – The DAG execution has been cancelled by the user
- *Cleared* – The Output Sandbox of all nodes has been transferred to the UI.

5.1 User-Level Commands for Workflow Management

A set of user commands is available, allowing users to submit and control the application workflow execution. The list of commands is as follows:

- edg-dag-submit – this command submits a DAG
- edg-job-status – this command checks the status of the submitted DAG
- edg-job-cancel – this permits a user to cancel the execution of a DAG
- edg-job-get-output – this command obtains output for all jobs of the DAG
- edg-job-get-logging-info – this presents logging info for a DAGMan execution.

6 Experimental Results

We conducted a number of experiments on parallel/distributed applications to study how this approach works in practice. We present a DAG whose structure is representative of the ATLAS (Atlas A Toroidal LHC ApparatuS) experiment [9] – the largest collaborative effort in the physical sciences. This experiment contains the following successive steps: event generation (evgen), simulation (sim), digitalization (digi). The first process does not take any input data, but rather generates certain output data. The second process processes data generated by the previous step, giving another data as result. The third process repeats the scheme of the sim step. To provide an efficient execution of the experiment, each step can be divided into N parts, where each part processes a subset of data as it is shown in Figure 3 (because of limits on space, we do not present the JDL specification of this DAG):

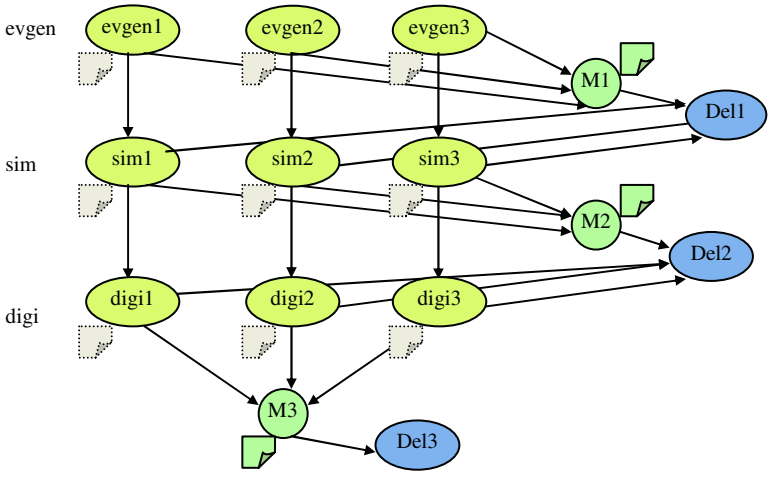


Fig. 3. An example DAG that represents the workflow of the ATLAS experiment

- evgen – this contains N sub-nodes, where each sub-node performs the partial event generation, and the generated partial result file is copied and registered in the Crossgrid storage using EDG Replica Manager [20].
- sim – this contains N sub-nodes, where each one copies the partial file generated by an evgen sub-node; it then performs the simulation on the partial data and generates a partial file. Finally, it copies and registers this file in the storage.
- digi – this contains N sub-nodes, where each one copies the partial file generated by a sim sub-node; it then performs digitalization on the partial data and generates a partial file. Finally, it copies and registers this file in the storage.

There is, additionally, a need for further steps that merge the temporal files generated by all parts of the considered step and remove such temporal files:

- Merge – this copies the files generated by all nodes (separately for each level) and merges them; it saves the results to a file and finally copies and registers this file in the storage; these merged files will be the result of the experiment.
- Delete – this deletes the files generated by all levels except Merge; this step can be carried out separately for each level.

Figure 4 presents an example execution of the workflow in the ATLAS experiment. The horizontal axis represents time, the vertical shows Computing Elements in which the workflow may be executed (Poland, Spain or Portugal). In this example, all nodes from the Event Generation level and merge operation have been successfully finished. During the execution of the simulation nodes, one of these has failed (sim2). This node has been retried automatically, but has failed again. It has not been possible to retry it as, e.g., a user specified `node_retry_count=1`. The workflow execution has terminated as failed. However, the Rescue DAG, which contains nodes annotated as already done, is provided for the user, who may therefore submit the failed workflow executing only those nodes that have not been successfully finished.

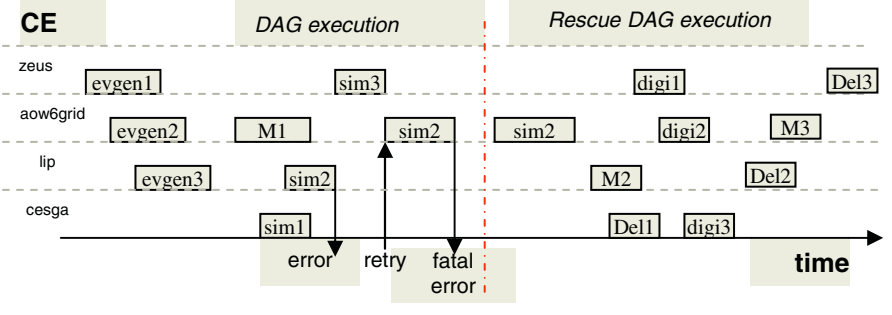


Fig. 4. An example execution of the DAG for the ATLAS experiment

7 Conclusions

The Grid system offers high computing capabilities for users in many scientific research fields. A great number of applications are made for a set of jobs that depend on each other. Generally, the execution of such applications in the grid environments requires users' intervention and the manual execution of each job, step by step, simulating their dependencies. It is therefore necessary to provide a good, reliable and simple service which automatically carries out the task of mapping the application workflow to the Grid.

In this paper, we have presented the solution to this problem, which has been applied within the EU-Crossgrid project. To provide workflow execution, we have represented such a workflow in the form of DAGs. The DAG execution is provided by the DAG Manager service and is integrated with the CrossGrid's Workload Management System (WMS). Our implementation is based on the DAGMan scheduler provided by the Condor group and is targeted to LCG-2 middleware.

Many aspects could be introduced to improve DAG support. For example, support for DAG in DAG (the possibility of specifying a node as another DAG) or the integration of DAG into the Migrating Desktop and Web Portal that provides a user-friendly interface by which to interact with the Grid.

References

1. I. Foster, C. Kesselman (Editors), "The GRID Blueprint for a New Computing Infrastructure". Morgan Kauffmann Publishers. 1999.
2. GriPhyN: The Grid Physics Network. http://www.griphyn.org
3. The DataGrid Project. http://www.eu-datagrid.org
4. GridLab: A Grid Application Toolkit and Testbed: http://www.gridlab.org
5. The EU-Crossgrid project. http://www.eu-crossgrid.org
6. W. Gropp, E. Lusk, N. Doss, A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard". Parallel Computing, 22(6), pp.789-828. 1996.
7. N. Karonis, B. Toonen, I. Foster, "MPICH-G2: A Gridenabled implementation of the message passing interface". Journal of Parallel and Distributed Computing, to appear. 2003.
8. E. Heymann, M.A. Senar, A. Fernandez, J. Salt, "The Eu-Crossgrid approach for Grid Application Scheduling". 1st European Across Grids Conference, LNCS series, pp.17-24. February, 2003.
9. B. Barish, R. Weiss, "Ligo and detection of gravitational waves". Physics Today, 52 (10). 1999.
10. S. Hastings, T. Kurc, S. Langella, U. Catalyurek, T. Pan, J. Saltz, "Image processing on the Grid: a toolkit or building grid-enabled image processing applications". In 3rd International Symposium on Cluster Computing and the Grid. 2003.
11. http://atlasexperiment.org/
12. http://www.cs.wisc.edu/condor/dagman/
13. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.H. Su, K. Vahi, M. Livny, "Pegasus : Mapping Scientific Workflows onto the Grid". Across Grids Conference. Nicosia, Cyprus, 2004.
14. M. Shields, "Programming Scientific and Distributed Workflow with Triana Services". GGF10 Workflow Workshop. Berlin, March, 2004.
15. UNICORE Plus - Final Report (2003). http://www.unicore.org
16. J. Cao, S.A. Jarvis, S. Saini, G.R. Nudd, "GridFlow: Workflow Management for Grid Computing". 3rd International Symposium on CCGrid. Japan, May 2003.
17. F. Pazini, "JDL Attributes - DataGrid-01-NOT-0101-0_4.pdf" http://www.infn.it/ workload-grid/docs/ DataGrid-01-NOT-0101-0_4-Note.pdf. December, 2001.
18. James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", Journal of Cluster Computing, vol. 5, pages 237-246, 2002.
19. Data grid: Definition of the architecture, technical plan and evaluation criteria for the resource coallocation framework and mechanisms for parallel job partitioning. WP1: Workload Management. DataGrid-01-D1.4-0127-1_0. Deliverable DataGrid-D1.4. 2002.
20. L. Guy, P. Kunszt, E. Laure, H. Stockinger, K. Stockinger, "Replica Management in Data Grids". Technical report, GGF5 Working Draft. July 2002.

Workflow-Oriented Collaborative Grid Portals[1]

Gergely Sipos[1], Gareth J. Lewis[2], Péter Kacsuk[1],
and Vassil N. Alexandrov[2]

[1] MTA SZTAKI Computer and Automation Research Institute,
H-1518 Budapest, P.O. Box 63, Hungary
{sipos, kacsuk}@sztaki.hu
[2] Advanced Computing and Emergent Technologies Centre,
School of Systems Engineering, University of Reading,
Whiteknights P.O. Box 225, Reading, RG6 6AY, UK
{v.n.alexandrov, g.j.lewis}@reading.ac.uk

Abstract. The paper presents how workflow-oriented, single-user Grid portals could be extended to meet the requirements of users with collaborative needs. Through collaborative Grid portals different research and engineering teams would be able to share knowledge and resources. At the same time the workflow concept assures that the shared knowledge and computational capacity is aggregated to achieve the high-level goals of the group. The paper discusses the different issues collaborative support requires from Grid portal environments during the different phases of the workflow-oriented development work. While in the design period the most important task of the portal is to provide consistent and fault tolerant data management, during the workflow execution it must act upon the security framework its back-end Grids are built on.

1 Introduction

The workflow concept is a widely accepted approach to compose large scale applications [14], [15], [16]. Most of today's well-known production Grids support the execution of workflows composed from sequential or parallel jobs [2], [3]. At the same time, none of these infrastructures contain services that enable the creation and processing of the workflows: the required functionality is provided by the front-end Grid portals. Globus 2 [1], the middleware layer the referred systems are built on, contains services for job-execution and for the different aspects of data management. Because workflow management builds directly onto these basic computation-related services and because the workflow support can be the top layer of a Grid, its integration into portals is an obvious simplification.

One of the original goals of grid computing is to enable the cooperative work between researcher or engineer teams through the sharing of resources [5]. In collaborative systems the different teams' software and hardware infrastructures and

[1] The work described in this paper is supported by the Hungarian Grid project (IHM 4671/1/2003) by the Hungarian OTKA project (No. T042459) and by the University of Reading.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 434 – 443, 2005.

the team member's knowledge can be aggregated to generate an environment that is suitable to solve previously unsolvable problems [6]. Unfortunately, today's different Grid portals focus solely on High Performance and High Throughput Computing, but they do not support collaborative work.

The P-GRADE Portal [4] is a Grid portal solution with such typical characteristics. On one hand it gives high-level workflow management support, on the other hand it is a useless piece of software for users with collaborative problem solving requirements. Using its integrated workflow editor component, or its graphical Web interface the users can comfortably organise sequential or parallel MPI jobs into workflows, can execute, monitor and visualize them, but cannot achieve the aggregation of knowledge, resources and results.

The main goal of our research was to determine how to convert the P-GRADE Portal, a Globus-based computational Grid portal into a centre for collaborative work. Although we discuss the problems collaborative computational portals face in reference to the P-GRADE Portal, the presented results can be easily generalised for other portal solutions like Triana [8] or GENIUS [9].

The remaining part of the paper is organised as follows. Chapter 2 discusses the several advantages workflow-oriented collaborative computational Grid portals bring for clients, while chapters 3 and 4 describe the different difficulties collaborative support requires in these environments. In chapter 3 the collaborative workflow design is introduced, while chapter 4 examines the differences between collaborative and single-user workflow execution processes. At the end chapter 5 gives conclusions.

2 Knowledge and Resource Aggregation with Collaborative Portals

Today's computational Grids sometimes seem quite small from the end-user's point of view. Even if a Grid has quite large number of sites an account or certificate is usually valid for a small subset of them, for the sites that participle in a specific Virtual Organisation (VO). Some VOs can be accessed by physicists, some others by geologists, while another subset is allocated for biologists. None of these scientists can break out from the "sandboxes" grid administrators force them into. This is true for the applications as well. Different VOs (or even Grids) are built for different jobs [3], and there is little chance for a program to run elsewhere than in its own special environment. Consequence of the usually special requirements of Grid jobs that the valuable results they produce bring benefit only for a few favoured researchers. Because Grid portals are designed to serve one specific VO (or sometimes more VOs inside the same Grid) there is no chance to define and execute across-grids workflows with them. The different jobs of such a workflow should run in different Grids (or at least in different VOs) and the portal framework must be able to handle the special design and execution needs such applications require.

With the collaborative P-GRADE Portal we would like to give solutions for this problem. The main aim of this new version of the P-GRADE Portal is to support the integration and sharing of knowledge and resources through collaborative workflows.

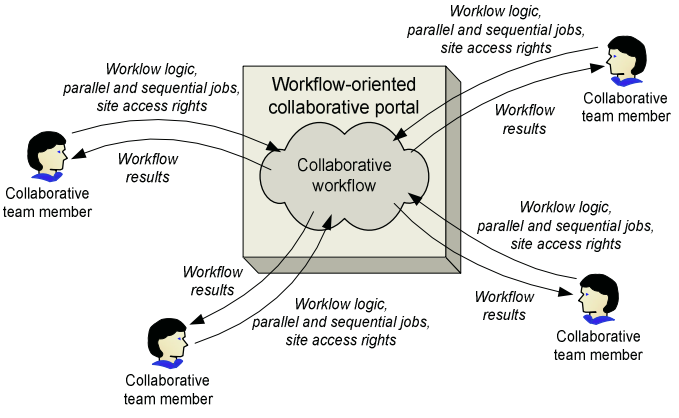


Fig. 1. Sharing and integrating knowledge and resources through collaborative portals

A collaborative workflow is a job-workflow built by multiple persons, by the members of a collaborative team. Every team member gives his/her own execution logic, jobs and resource access rights to the workflow graph, enabling the team to exploit aggregated knowledge and hardware capacity. This main goal of collaborative portals can be seen in Fig. 1.

If we first take a look at a single-user P-GRADE Portal workflow, it can be stated that one individual must possess all the knowledge and site access rights the workflow definition and execution requires [4]. The person that defines the graph must specify the jobs for the different nodes and must have a certificate which is valid to execute these jobs on the chosen Grid sites. (Even if a high-level broker service automatically maps the jobs onto the best sites it is the consequence of GSI that the end-user must possess with the appropriate certificate(s) that are accepted by those sites.) This it true for all the previously referred, Globus 2 based Grids and their portals.

Using the team work support of a collaborative portal, the different participants can aggregate their knowledge and abilities granted them by their certificates to achieve higher level goals than they individually would be able to. In the design phase the team members can collaboratively and in real-time define the structure of the workflow graph and the content of the different nodes. Every member can contribute to the workflow with his/her jobs. A job can use the results produced by other members' jobs and can generate results for other jobs again. Through the data flow between the jobs inside a collaborative workflow the system realises the flow of knowledge. The different users do not have to take care where and how the input files for their jobs will be produced, they have to know only the format and the meaning of the incoming and outgoing data flows.

Globus 2 and Globus 3 based Grids can be accessed through the P-GRADE Portal [4]. In Globus 2 environments GRAM sites provide the job execution facility and consequence of the GSI philosophy is that different users can access different GRAM sites. These sites can be inside the same VO, inside different VOs within the same Grid, or in totally different Grids. The P-GRADE Portal does not cooperate with

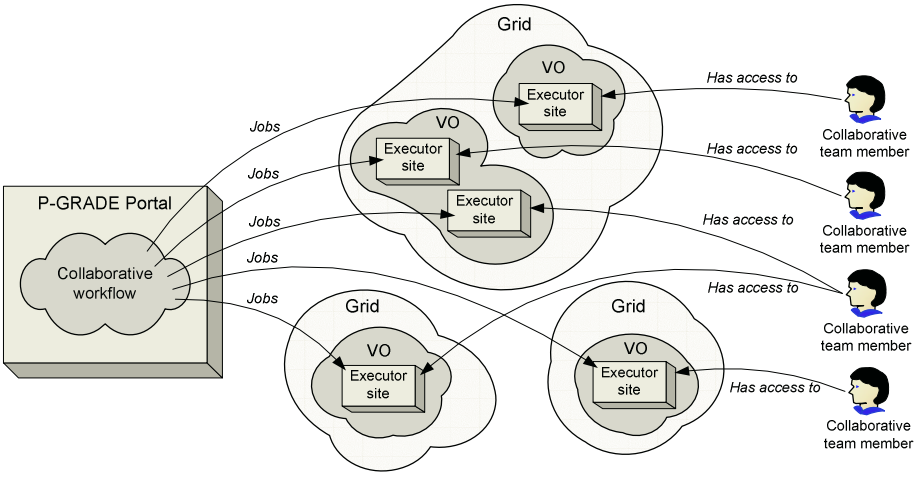


Fig. 2. Collaborative inter-grid resource aggregation

Grid resource brokers, it sends the different jobs to GRAM sites statically defined by the developers of the workflows. Since several participants are involved in the construction of a collaborative workflow, the sites they can individually access are collectively available for the collaborative jobs. Any job of a collaborative workflow can use any site that at least one team member has access to. (See Fig. 2.) Due to this philosophy workflow-based inter-grid resource aggregation can be realised.

3 Collaborative Workflow Development

The P-GRADE Portal provides a dynamically downloadable editor application what the users can apply to construct and upload workflows onto the portal server [4]. A P-GRADE workflow consists of three types of components: jobs, ports and links. Jobs are the sequential programs and parallel MPI applications that must be executed on the different Grid sites. Ports are the output and input files these jobs generate or require, while links define data channels between them.

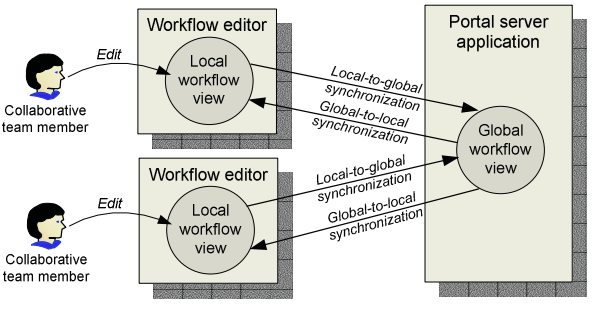


Fig. 3. The different synchronization tasks in the collaborative workflow editing period

In the collaborative version of the P-GRADE portal multiple editors must be able to work on the same workflow definition simultaneously. To enable the parallel work, every editor must manage its own local copy of the actually opened collaborative workflow and synchronize it with the appropriate global view stored on the portal server. The purpose of the synchronization is twofold: with a local-to-global update the locally performed changes can be validated on the consistent global view, while the global-to-local update is necessary to inform the different team members about each others' work in a real-time fashion. The different synchronization tasks are illustrated in Fig. 3.

The server component of the P-GRADE Portal is a Web application that cannot send asynchronous calls to its client side workflow editors [4]. Consequently, both the global-to-local and the local-to-global synchronization processes have to be initialized by the clients. During a local-to-global synchronization process the portal server has to generate a new global view from the present global and the received local workflows. During a global-to-local synchronization the workflow editor has to generate a new local view from the present local and the received global workflows.

The most obvious way to update a workflow with another one is to simply overwrite it with the new one. It can be clearly seen that this protocol would lead to lost updates and would make the final result of the editing phase unpredictable: the user, whose editor updates the global workflow last, overwrites the other members' already validated work. Database manager systems successfully solve similar problems with data locking mechanisms [10]. These servers lock smaller or bigger parts of their managed data to provide consistent view. In our case the global workflow is the managed data. Because locking of full workflows would prevent the collaborative work itself, (in that case only one person could work on a workflow at a time) collaborative portals have to enable the locking of workflow parts. A workflow part can be locked for maximum one user at a time, but different parts of the same workflow can be locked for different users simultaneously. Only locked workflow parts can be edited and only by their owners. On the other hand every collaborative group member must be able to read both locked and unlocked parts of the collaborative workflow.

Our approach cuts workflow design periods to three phases: contention, editing and releasing. In the contention phase the user must lock the workflow part he/she is interested in. In the editing phase the locked part can be modified, while in the releasing phase the modified components get unlocked and become part of the consistent global view.

In our proposed concept locking requests can be generated through the workflow editor by the users manually. When a collaborative team member clicks a special menu item of a workflow node his/her editor generates a locking request. The meaning of this request for the system is the following: take the part of the workflow graph which starts with the selected job, and lock as big continuous part of it as possible. In other words the system has to lock the selected job, every job that directly or indirectly depends on this job via file dependencies, all ports that are connected to these jobs, and finally all direct links between these jobs. If a locked component is found during this graph traversal process, the system skips the rest of the branch and continues the work on the remaining branches. Fig. 4. illustrates the protocol through a simple example.

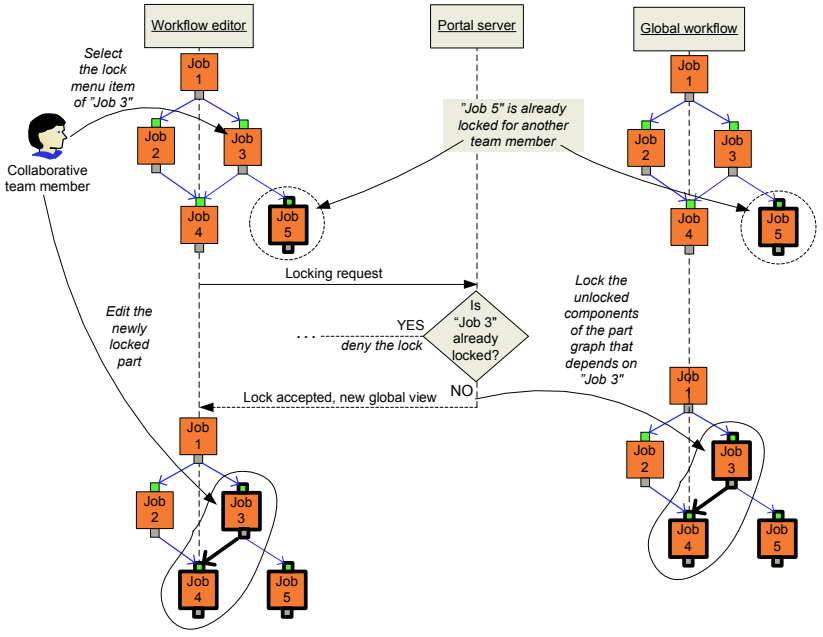


Fig. 4. Workflow component locking scenario

The extended sequence diagram presents a locking scenario. On the left side one member of a collaborative team can be seen. He participates in the collaborative design through his workflow editor. The editor manages the local view of the collaborative workflow, and it communicates with the portal server that stores the global view. At the beginning both the editor and the portal server store the same view: “Job 5” together with its input and output ports (the small squares connected to it) are locked, while the rest of the workflow is unlocked. This means that some other person from the group is currently working on “Job 5”, but the rest of the graph is free to lock.

Suppose that our user clicks the “Lock” menu of “Job 3”. The editor automatically generates an appropriate request message and sends it to the server. The server appoints that the status of “Job 3” is unlocked, so it estimates the biggest part of the graph that depends on “Job 3” and contains only unlocked components. In our case this part graph consists of “Job 3”, “Job 4”, their two input and output ports, and the link that connects them. (Since “Job 5” was already locked it does not become part of this group.) The server locks this branch in the global view and sends this new view back to the editor. The editor updates its GUI according to the received graph allowing the user to begin the development work. The user now can modify the properties of his/her components, he/she can add new jobs to this graph, can connect new ports to his jobs, can define new links between his ports and finally he can delete any of his locked components.

The development phase normally comes to its end when the user finishes the work and manually unlocks the components just like he/she locked them earlier. Besides this the system has to be prepared for unstable clients as well. The workflow editor can crash, the client host or the network can break down, hence the portal server must be able to identify and automatically unlock unnecessarily locked workflow components. Distributed systems apply leasing/lifetime based solutions for this purpose [11], [12]. If the portal server locks workflow components only for limited periods and enables the on-demand extensions of these locking intervals then unnecessarily locked components can be released as soon as possible. The workflow editor can hide the regular task of the leasing/lifetime extension from the user.

The workflow editor has to perform regular local-to-global synchronization (see Fig. 3.) during the whole design period. In the contention phase out-of-date local information can make the user believe that a given part of the workflow is still locked by someone else, or contrarily, it is still free to lock. In the editing phase rarely updated local views deprive the users from the experience of the real-time collaboration. During a global-to-local synchronization process the editor has to merge the received global workflow with the present local one. The new local view must be generated from the following items:

- every component from the old local view that is locked for the local user
- every unlocked component from the received global view
- every component from the global view that is locked for other users

Since only locked components can be modified, the local-to-global synchronization process makes no sense in the contention phase. During a local-to-global synchronization process the portal server has to merge the present global workflow with the received local one. The following items have to be aggregated to generate the new global view:

- every component from the local view that is locked for the user whose editor performed the local-to-global update process
- every component from the global view that is locked for other users
- every unlocked component from the global view.

4 Collaborative Workflow Execution

In the execution phase there are two differences between collaborative and single-user Globus workflows:

1. In the collaborative case multiple clients must be able to observe and control the execution of a workflow.
2. To execute a collaborative workflow usually more than one GSI proxies are required. For a normal workflow a single proxy is enough.

The synchronization process discussed earlier provides an adequate solution to the first issue. The system can provide a consistent view during in the execution phase just like it is achieved in the editing period.

Our proposed solution for the second problem is not so simple. The present version of the P-GRADE Portal generates a Globus proxy for a workflow job from the long-

term certificate that belongs to the person who defined GRAM site for that job [4]. To enhance security, the P-GRADE Portal delegates the certificate and proxy management tasks to MyProxy servers [13]. The Portal itself is only a Web interface to access MyProxy sites. The users can upload and download certificates and proxies between their client machines, the portal server and the MyProxy hosts through their Web browsers. Collaborative portals must be able to automatically download proxies for the different collaborative jobs, since a user, who submits a collaborative workflow cannot wait until all the involved team members download the necessary proxies by hand.

Collaborative portals can provide an automatic proxy download facility if they store the relations between users, MyProxy sites, and GRAM hosts. Using this database the portal can estimate which MyProxy server must be used to get a valid proxy for a GRAM site defined by a collaborative group member. The infrastructure required to process the automated proxy download can be seen in Fig. 5. The huge benefit of this approach is that once the collaborative group members define their own MyProxy-GRAM relations, the portal can always get the required proxies without any manual help.

Another consequence of the usage of multiple proxies within a single workflow is that file transfers between different security domains necessarily occur. In this case none of the proxies is sufficient to perform direct file transfer between the job executor sites. The portals server has to be used as a temporary storage for the indirect file transfer. The portal server application can use the proxy of the first job to copy the output files into a permanent or temporary directory on the portal server, and it can use the proxy of the second job to copy these files onto the second GRAM site. After the successful finish of the workflow the proxies that belong to the jobs that produced the final results can be used to copy the files onto the portal server at a place where every collaborative team member can access them.

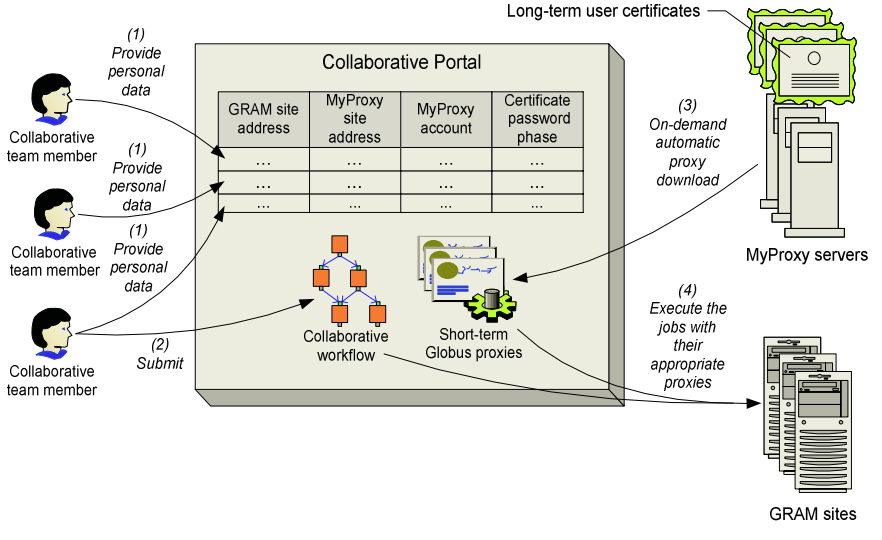


Fig. 5. Infrastructure for automated execution of collaborative workflows

5 Summary and Conclusions

The paper discussed how today's Globus-based, workflow-oriented computational Grid portals can be adapted to collaborative centres. The solutions have been examined through the example of P-GRADE Portal, a concrete computational Grid portal, but most of the results can be applied for other widely used frameworks as well.

In the workflow design phase the biggest difference between single-user and collaborative portals is the requirement to protect against lost updates. Our approach is a data locking mechanism optimised for workflow-oriented environments. In the execution phase security is the biggest issue for collaborative groups. The paper introduced an automatically proxy download facility to submit the workflow jobs into the different VOs and Grids the collaborative team members have access to. The solution is based on a special database that specifies which proxies and how must be obtained to access the specified executor sites.

Because of the extra features that collaborative portals must provide in contrast with single-user portals, these complex systems loose their interface characteristics and become functionality providers. Since Grid portals should be slim layers that provide Web interface for standardised Grid services, the introduced collaborative functionality should be realised by stateful Grid services and not by a Grid portal. Jini [11] or WSRF [21] would be possible nominates to develop these services.

The P-GRADE Portal is already used in several Grid systems like the LCG-2 based SEE-GRID [18], the GridLAB testbed [8], the Hungarian Supercomputing Grid [19] and the UK OGSA testbed [20]. We already started the implementation of its collaborative version. The prototype will be available by June 2005 and the final system will be absolutely unique on the market and will provide an optimal balance between HPC, HTC and collaborative support.

References

[1] Foster and C. Kesselman: "The Globus project: A status report", In Proceedings of the Heterogeneous Computing Workshop, pages 4-18. IEEE Computer Society Press, 1998.

[2] The Grid2003 Production Grid: "Principles and Practice", To appear in 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC13), Honolulu, 2004.

[3] LHC Grid: http://lcg.web.cern.ch/LCG/

[4] Cs. Németh, G. Dózsa, R. Lovas and P. Kacsuk: The P-GRADE Grid portal In: Computational Science and Its Applications – ICCSA 2004: International Conference, Assisi, Italy, LNCS 3044, pp. 10-19

[5] Foster and C. Kesselman: "The Grid: Blueprint for a New Computing Infrastructure", Morgan-Kaufman, 1999.

[6] Gareth J. Lewis, S. Mehmood Hasan, Vassil N. Alexandrov: "Building Collaborative Environments for Advanced Computing" In proc. of the 17th International Conference on Parallel and Distributed Systems (ISCA), pp. 497-502, San Francisco, 2004.

[7] Foster, C. Kesselman, G. Tsudik, and S. Tuecke: "A security architecture for computational grids", In ACM Conference on Computers and Security, pages 83–91. ACM Press, 1998.

[8] G. Allen, K. Davis, K. N. Dolkas, N. D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor: "Enabling Applications on the Grid: A GridLab Overview", International Journal of High Performance Computing Applications, Aug. 2003.

[9] R. Barbera, A. Falzone, A. Rodolico: "The GENIUS Grid Portal", Computing in High Energy and Nuclear Physics, 24-28 March 2003, La Jolla, California

[10] V. Gottemukkala and T. Lehman: "Locking and latching in a memory-resident database system", In Proceedings of the Eighteenth International Conference on Very Large Databases, Vancouver, pp. 533-544, August 1992.

[11] J. Waldo: "The Jini architecture for network-centric computing", Communications of the ACM, 42(7), pp. 76-82, 1999.

[12] Foster, C. Kesselman, J. Nick, and S. Tuecke: "The physiology of the Grid: An Open Grid Services Architecture for distributed systems integration", Technical report, Open Grid Services Architecture WG, Global Grid Forum, 2002.

[13] J. Novotny, S. Tuecke, and V. Welch: "An online credential repository for the grid: MyProxy", Symposium on High Performance Distributed Computing, San Francisco, Aug. 2001.

[14] Taylor, M. Shields, I. Wang and R. Philp: "Grid Enabling Applications Using Triana", In Workshop on Grid Applications and Programming Tools. Seattle, 2003.

[15] Matthew Addis, et al: "Experiences with eScience workflow specification and enactment in bioinformatics", Proceedings of UK e-Science All Hands Meeting 2003

[16] Ewa Deelman, et al: "Mapping Abstract Complex Workflows onto Grid Environments", Journal of Grid Computing, Vol.1, no. 1, 2003, pp. 25-39.

[17] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke: "Condor-G: A Computation Management Agent for Multi-Institutional Grids", in 10th International Symposium on High Performance Distributed Computing. IEE Press, 2001

[18] SEE-GRID Infrastructure: http://www.see-grid.org/

[19] J. Patvarszki, G. Dozsa, P Kacsuk: "The Hungarian Supercomputing Grid in the actual practice", Proc. of the XXVII. MIPRO Conference, Hypermedia and Grid Systems, Opatija, Croatia, 2004. pp. 203-207.

[20] T. Delaitre, A.Goyeneche, T.Kiss, G.Z. Terstyanszky, N. Weingarten, P. Maselino, A. Gourgoulis, S.C. Winter: "Traffic Simulation in P-Grade as a Grid Service", Proc. of the DAPSYS 2004 Conference, September 19-22, 2004, Budapest, Hungary

[21] Foster, I., J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana: "Modeling Stateful Resources with Web Services", 2004, www.globus.org/wsrf

Contextualised Workflow Execution in MyGrid

M. Nedim Alpdemir[1], Arijit Mukherjee[2], Norman W. Paton[1], Alvaro A.A. Fernandes[1], Paul Watson[2], Kevin Glover[3], Chris Greenhalgh[3], Tom Oinn[4], and Hannah Tipney[1]

[1] Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, United Kingdom
[2] School of Computing Science, University of Newcastle upon Tyne, Newcastle upon Tyne NE1 7RU, United Kingdom
[3] School of Comp. Sci. and Inf. Tech., University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK
[4] European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, United Kingdom

Abstract. e-Scientists stand to benefit from tools and environments that either hide, or help to manage, the inherent complexity involved in accessing and making concerted use of the diverse resources that might be used as part of an *in silico* experiment. This paper illustrates the benefits that derive from the provision of integrated access to contextual information that links the phases of a problem-solving activity, so that the steps of a solution do not happen in isolation, but rather as the components of a coherent whole. Experiences with myGrid workflow execution environment (Taverna) are presented, where an information model provides the conceptual basis for contextualisation. This information model describes key characteristics that are shared by many e-Science activities, and is used both to organise the scientist's personal data resources, and to support data sharing and capture within the myGrid environment.

1 Introduction and Related Work

Grid-based solutions to typical e-Science problems require the integration of many distributed resources, and the orchestration of diverse analysis services in a semantically rich, collaborative environment [5]. In such a context, it is important that e-Scientists are supported in their day-to-day experiments with tools and environments that allow the principal focus to be on scientific challenges, rather than on the management and organisation of computational activities.

Research into Problem Solving Environments (PSEs) has long targeted this particular challenge. Although the term *Problem Solving Environment* means different things to different people [4], and its meaning seems to have been evolving over time, a number of common concepts can be identified from the relevant research (e.g. [4, 6, 12, 8]). For example, the following features are commonly supported: problem definition; solution formulation; execution of the problem solution; provenance recording while applying the solution; result visualisation and

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 444–453, 2005.

analysis; and support for communicating results to others (i.e. collaboration). Although these are among the most common features, different PSEs add various other capabilities, such as intelligent support for problem formulation and solution selection, or highlight a particular feature, such as the use of workflow (e.g. [3, 1, 11]).

This paper emphasizes a specific aspect that has been largely overlooked, namely the provision of integrated access to *contextual information* that links the phases of a problem-solving exercise in a meaningful way. In myGrid, the following are principles underpin support for contextualisation:

Consistent Representation: The information model ensures that information required to establish the execution context conforms to a well-defined data model, and therefore is understood by the myGrid components that take part in an experiment as well as external parties.

Automatic Capture: When a workflow is executed, contextual information is preserved by the workflow enactment engine, and used to annotate both intermediate and final results.

Long-term preservation: The contextual information used to organise the persistent storage of provenance information on workflows and their results, easing interpretation and sharing.

Uniform Identification: Both contextual and experimental data are identified and linked using a standard data and metadata identification scheme, namely LSIDs [2].

The rest of the paper describes contextualisation in myGrid, indicating both how contextualisation supports users and how the myGrid architecture captures and conveys the relevant information. As such, Section 2 summarises the information model, which is at the heart of contextualisation. Section 3 provides some concrete examples of contextualisation in practice, in a bioinformatics application. Section 4 provides an architectural view of the execution environment, and finally Section 5 presents some conclusions.

2 The Information Model

The myGrid project (`http://www.mygrid.org.uk/`) is developing high-level middleware to support the e-Scientist in conducting *in silico* experiments in biology. An important part of this has been the design of an Information Model (IM) [9], which defines the basic concepts through which different aspects of an e-Science process can be represented and linked. By providing shared data abstractions that underpin important service interactions, the IM promotes synergy between myGrid components. The IM is defined in UML, and equivalent XML Schema definitions have been derived from the UML to facilitate the design of the myGrid service interfaces.

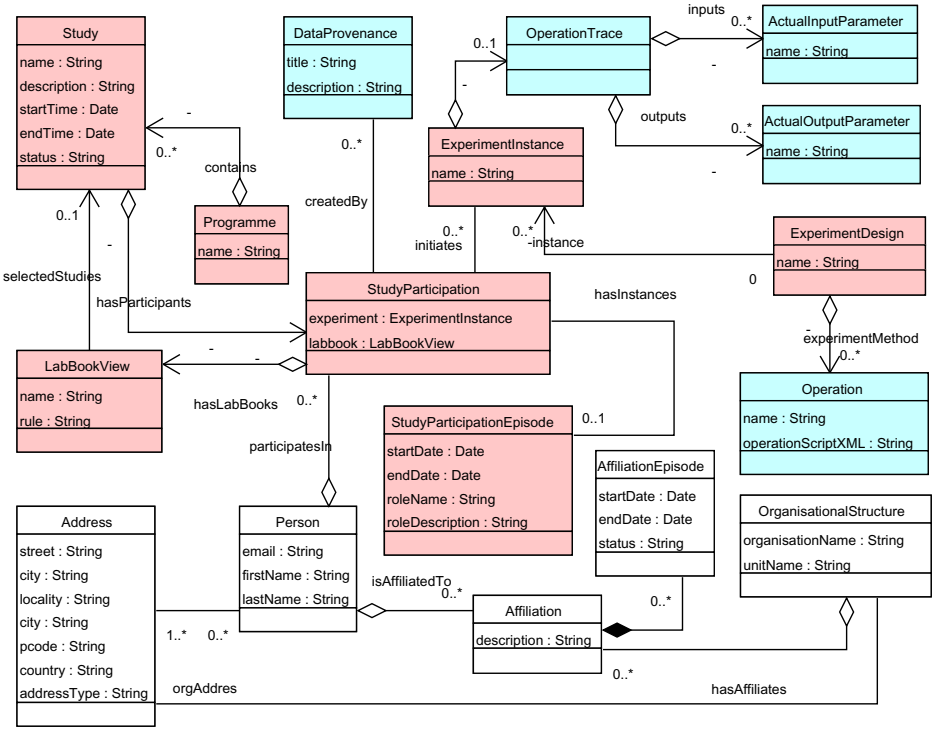


Fig. 1. A UML class diagram providing an overview of the information model

Figure 1 illustrates several of the principal classes and associations in the IM. In summary; a *Programme* is a structuring device for grouping other *Studies* and can be used to represent e.g. a project or sub-project. An *Experiment Design* represents the method to be used (typically as a workflow script) to solve a scientific problem. An *Experiment Instance* is an application of an *Experiment Design* and represents some executing or completed task. The relationship of a *Person* with an *Organizational Structure* is captured by an *Affiliation*. A *Study Participation* qualifies a person's relationship to the study by a set of study roles. An *Operation Trace* represents inputs, outputs and the intermediate results of an experiment (i.e. the *experiment provenance*), as opposed to the *Data Provenance* which primarily indicates a data item's creation method and time.

An important feature of the IM is that it does not model application-specific data, but rather treats such data as opaque, and delegates responsibility for its interpretation to users and to application-specific services. As such, concepts such as *sequence* or *gene*, although they are relevant to the Williams-Beuren Syndrome (WBS) case study described in Section 3.1, are not explicitly described in the IM. Rather, the IM captures information that is common to, and may even be shared by, many e-Science applications, such as scientists, studies and

experiments. A consequence of this design decision is that the myGrid components are less coupled to each other and to a particular domain, and so are more easily deployable in different contexts. However, interpretation and processing (e.g. content aware storage and visualisation) of the results for the end user becomes more difficult, and falls largely on the application developer's shoulders.

3 Contextualized Workflows: A User's Perspective

3.1 Case Study

Informatic studies in Williams-Beuren Syndrome (WBS) are used to illustrate the added value obtained from contextualisation. WBS is a rare disorder characterized by physical and developmental problems. The search for improved understanding of the genetic basis for WBS requires repeated application of a range of standard bioinformatics techniques. Due to the highly repetitive nature of the genome sequence flanking the Williams-Beuren syndrome critical region, sequencing of the region is incomplete, leaving documented gaps in the released genomic sequence. In order to produce a complete and accurate map of the region, researchers must constantly search for newly sequenced human DNA clones that extended into these gap regions [10].

Several requirements of the WBS application stand to benefit from integrated support for contextualisation:

- The experimenter needs to conduct several tasks repeatedly (e.g. execution of follow-on workflows), which requires the inputs, outputs and intermediate results of one step to be kept in the same experimental context, to ease comparisons of multiple runs and of alternative approaches.
- Results or experimental procedures need to be shared among researchers in the group. A contextualized environment helps scientists to migrate from ad-hoc practices for capturing the processes followed and the results obtained, to computer-supported information-rich collaboration schemes.

3.2 Contextual Data in Use

This section illustrates how integrated support for contextualisation surfaces to the user. In myGrid, workflows are developed and executed using the Taverna workbench [7], which is essentially a workflow editor and a front-end to a workflow execution environment with an extensible architecture, into which additional components can be plugged. The following myGrid plug-ins provide users with access to external information resources when designing and executing workflows:

MIR Browser: The myGrid Information Repository (MIR) is a web service that provides long-term storage of information model and associated application-specific data. The plug-in supports access to and modification of data in the MIR.

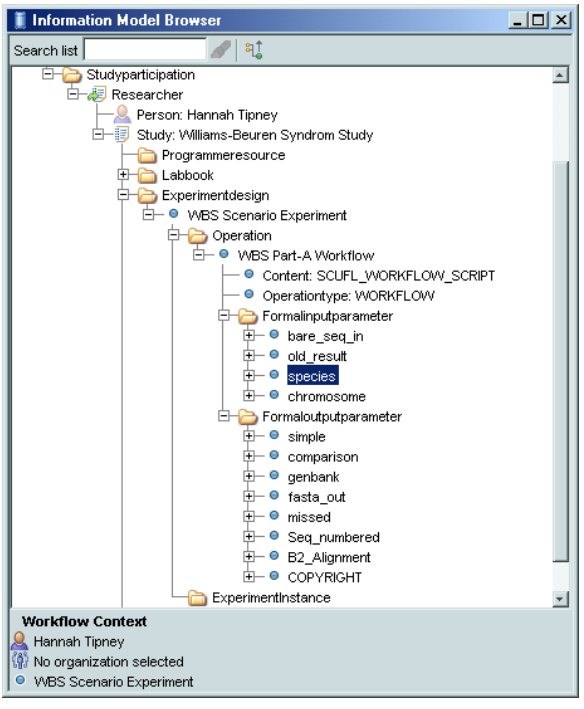


Fig. 2. MIR Browser displaying context information

Metadata Browser: The myGrid Metadata Store is a web service that supports application-specific annotations of data, including data in the MIR, using semantic-web technologies. The plug-in supports searching and browsing of information in the metadata store, as well of the addition of new annotations.

Feta Search Engine: The Feta Search Engine provides access to registry data on available services and resources.

Users are supported in providing, managing or accessing contextual data using one or more of the above plug-ins, and benefit from the automatic maintenance of contextual data in the MIR.

When developing or using workflows, the e-Scientist first launches the Taverna workbench, which provides workflow-specific user interface elements, as well as the plug-ins described above. When the MIR browser is launched, the user provides login details, and is then provided with access to their personal instance of the information model, as illustrated in Figure 2. This figure shows how access has been provided, among other things, to: (i) data from the studies in which the e-Scientist is participating – in this case a *Williams-Beuren Syndrome Study*; (ii) the experiment designs that are being used in the study – in this case a single *WBS-Scenario Experiment*; and (iii) the definitions of the workflows

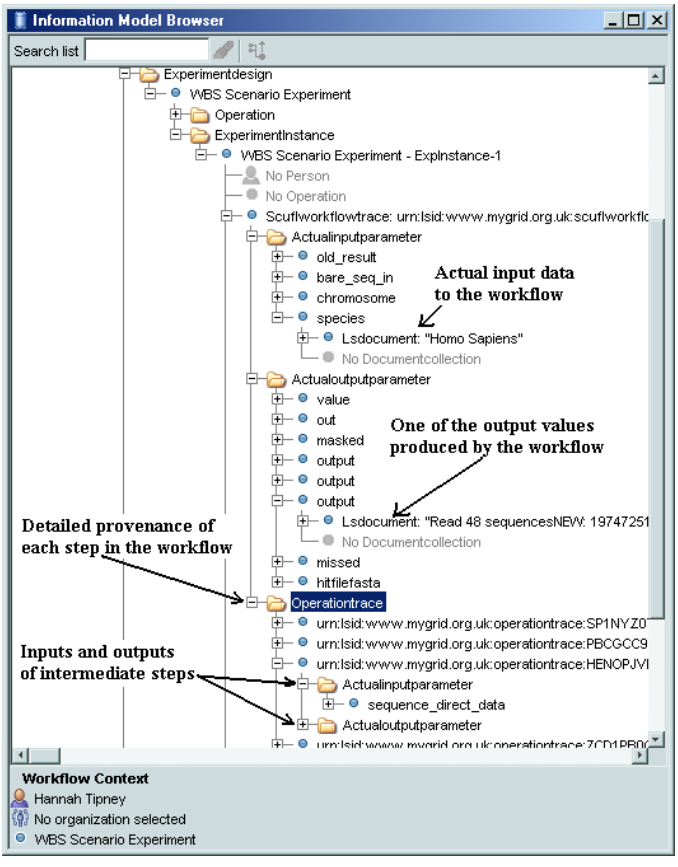


Fig. 3. Workflow execution results in the MIR Browser

that are used in the *in silico* experiments – in this case a single workflow named *WBS part-A workflow*. In this particular case, the absence of a + beside the *ExperimentInstance* indicates that the *WBS-Scenario Experiment* has not yet been executed. At this point, the e-Scientist is free either to select the existing workflow, or to search for a new workflow using Feta search engine. Either way, it is possible for the workflow to be edited, for example by adding new services discovered using Feta to try variations to an existing solution.

When a workflow is selected for execution, the e-Scientist can obtain data values for use as inputs to the workflow from previous experiment results stored in the MIR. The execution of follow-on analyses is common practice in the WBS case study.

The user's view in Figure 2 illustrates that resources, such as individual workflows, do not exist in isolation, but rather are part of a context – in this case a study into WBS. There may be many experiments and workflows that are part of that study. In addition, when a workflow from a study is executed, it is executed in the context of that study, and the results of the workflow are

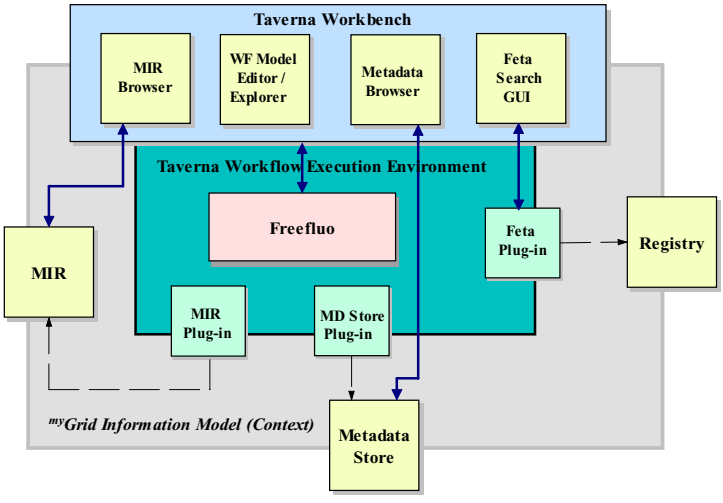


Fig. 4. A simplified architectural view

automatically considered to be among the results of the study. For example, Figure 3 illustrates the results obtained by executing *WBS part-A workflow* from Figure 2. The results of the execution are automatically recorded under the *Experiment Instance* entity, which is associated with the *Experiment Design* from Figure 2. The values obtained for each of the *Formaloutputparameter* values from Figure 2 are made available as *Actualoutputparameter* values in Figure 3. In addition, provenance information about the workflow execution has also been captured automatically. For example, the LSID [2] of each operation invoked from the workflow is made available as an *Operationtrace* value. Further browsing of the individual operation invocations indicates exactly what values were used as input and output parameters. Such provenance information can be useful in helping to explain and interpret the results of *in silico* experiments.

4 Contextualised Workflows: An Architectural Perspective

The core components that participate in the process of formulating and executing a workflow were introduced in Section 3.2. Figure 4 illustrates principal relationships between the components, where the Taverna workbench constitutes the presentation layer, and includes a number of GUI plug-ins to facilitate user interaction. The workflow enactor (Freefluo) is the central component of the execution environment, and communicates with other myGrid components via plug-ins that observe the events generated as the enactor's internal state changes. For example, when an intermediate step in the workflow completes its execution, the enactor generates an event and makes the intermediate results available to the event listeners. The MIR plug-in responds to this event by obtaining the

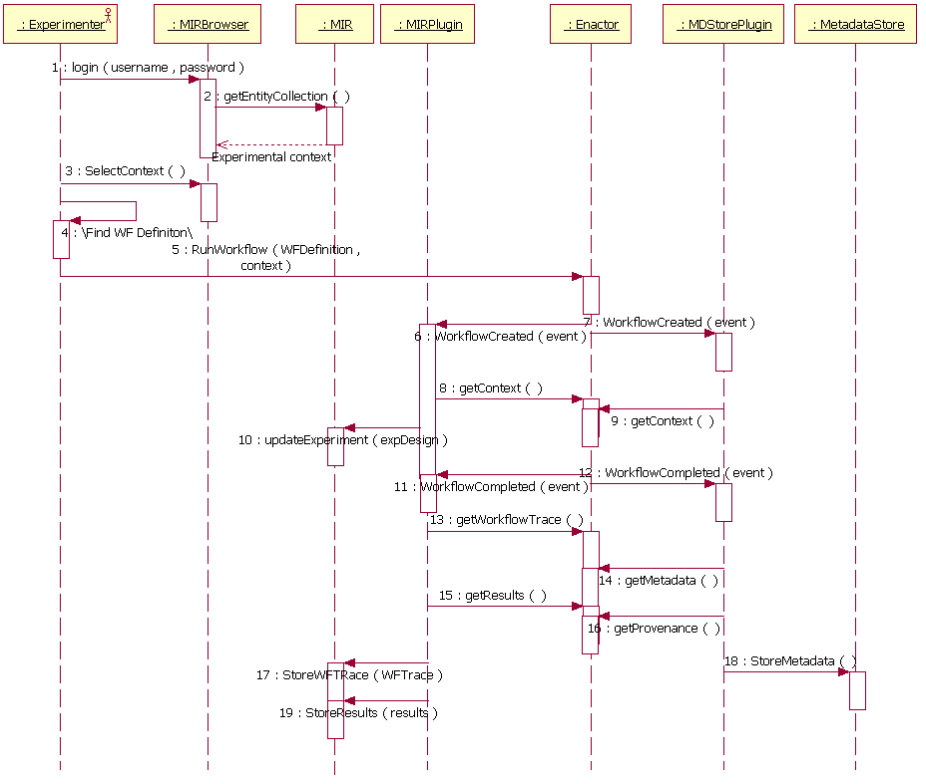


Fig. 5. Interactions between core myGrid components

intermediate results and storing them in the MIR in their appropriate context. As such, the plug-in architecture is instrumental in facilitating the automatic propagation of the experimental context across the participating components.

Figure 5 is a UML sequence diagram that illustrates a basic set of interactions between the myGrid components from in Figure 4, and provides a simplified view of the steps involved in contextualised execution of a workflow. A typical interaction for a contextualized workflow execution starts with user's login via the MIRBrowser. The next step is normally finding a workflow to execute. This could either be done by a simple load from the local file system, by a get operation from the MIR, or by a search query via the Feta GUI panel. Next, the experimenter executes the workflow. Although in the diagram this is shown as a direct interaction with the enactor for simplicity, in reality this is done via a separate GUI panel and the context is passed to the enactor implicitly. As the enactor executes the workflow, it informs event listeners (i.e. the MIR plug-in and the Metadata Store plug-in) that act as proxies on behalf of other myGrid components, at each critical event. Only two important events, namely *WorkflowCreated* and *WorkflowCompleted*, are shown in the diagram, although there

are several other intermediate events emitted by the enactor, for example for capturing provenance data on operation calls. The listeners respond to those events by extracting the context information and any other information they need from the enactor's state, thereby ensuring that the MIR and the Metadata Store receive the relevant provenance information.

An additional benefit of the automatic capturing of workflow results and provenance information is that the e-Scientist can pose complex queries against historical records. For example, a query could be expressed to *select all the workflows that were executed after date 30th March 2004, by the person 'Hannah Tipney', that had an output of type 'BLAST output'.*

5 Conclusions

This paper has described how a workflow definition and enactment environment can provide enhanced support for e-Science activities by closely associating workflows with their broader context. The particular benefits that have been obtained in myGrid derive from the principles introduced in Section 1, namely:

Consistent Representation: the paper has described how an e-Science specific, but application-independent, information model can be used not only to manage the data resources associated with a study, but also to drive interface components. In addition many myGrid components take and return values that conform to the information model, leading to more consistent interfaces and more efficient development.

Automatic Capture: the paper has described how the results of a workflow execution, plus associated provenance information, can be captured automatically, and made available throughout an e-Science infrastructure as events to which different components may subscribe. The properties of these events are generally modelled using the information model, and are used in the core myGrid services to support the updating of displays and the automatic storage of contextualised information.

Long-term Preservation: the paper has described how an information repository can be used to store the data artifacts of individual scientists in a consistent fashion, thereby supporting future interpretation, sharing and analysis of the data. Most current bioinformatics analyses are conducted in environments in which the user rather than the system has responsibility for recording precisely what tasks have taken place, and how specific derived values have been obtained.

Uniform Identification: the paper has described how a wide range of different kinds of data can provide useful context for the conducting of *in silico* experiments. Such information often has to be shared, or cross-referenced. In myGrid, LSIDs are used to identify the different kinds of data stored in the MIR, and LSIDs also enable cross-referencing between stores. For example, if an assertion is made about a workflow from an MIR in a Metadata Store, the Metadata Store will refer to the workflow by way of its LSID.

As such, the contribution of this paper has been both to demonstrate the benefits of contextualisation for workflow enactment, and also to describe the myGrid approach, both from a user and architectural perspective. The software described in this paper is available from *http://www.mygrid.org.uk*.

Acknowledgements. The work reported in this paper has been supported by the UK e-Science Programme.

References

1. S. AlSairafi et al. The design of Discovery Net: Towards Open Grid Services for Knowledge Discovery. *The International Journal of High Performance Computing Applications*, 17(3):297 – 315, Fall 2003.
2. T. Clark, S. Martin, and T. Liefeld. Globally distributed object identification for biological knowledgebases. *Briefings in Bioinformatics*, 5(1):59 – 70, 2004.
3. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus : Mapping scientific workflows onto the grid. In I. Foster and C. Kesselman, editors, *2nd European Across Grids Conference*, 2004.
4. E. Gallopoulos, E. Houstis, and J. R. Rice. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Comput. Sci. Eng.*, 1(2):11–23, 1994.
5. C. Goble, C. Greenhalgh, S. Pettifer, and R. Stevens. Knowledge integration: In silico experiments in bioinformatics. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 121–134. Morgan Kaufmann, 2004.
6. E. N. Houstis and J. R. Rice. Future problem solving environments for computational science. *Math. Comput. Simul.*, 54(4-5):243–257, 2000.
7. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, page bth361, 2004.
8. K. Schuchardt, B. Didier, and G. Black. Ecce – a problem-solving environment's evolution toward grid services and a web architecture. *Concurrency and Computation: Practice and Experience*, 14(13 – 15):1221 – 1239, 2002.
9. N. Sharman, N. Alpdemir, J. Ferris, M. Greenwood, P. Li, and C. Wroe. The myGrid Information Model. In S. J. Cox, editor, *Proceedings of UK e-Science All Hands Meeting 2004*. EPSRC, September 2004.
10. R. D. Stevens, H. J. Tipney, C. J. Wroe, T. M. Oinn, M. Senger, P. W. Lord, C. A. Goble, A. Brass, and M. Tassabehji. Exploring Williams-Beuren syndrome using myGrid. *Bioinformatics*, 20(suppl_1):i303–310, 2004.
11. I. Taylor, M. Shields, and I. Wang. *Grid Resource Management*, chapter Resource Management of Triana P2P Services. Kluwer, June 2003.
12. D. W. Walker, M. Li, O. F. Rana, M. S. Shields, and Y. Huang. The software architecture of a distributed problem-solving environment. *Concurrency: Practice and Experience*, 12(15):1455–1480, 2000.

Real World Workflow Applications in the Askalon Grid Environment*

Rubing Duan, Thomas Fahringer, Radu Prodan, Jun Qin, Alex Villazón, and Marek Wieczorek

Institute for Computer Science, University of Innsbruck,
Technikerstraße 21a, A-6020 Innsbruck, Austria
{rubing,tf,radu,jerry,alex,marek}@dps.uibk.ac.at

Abstract. The workflow paradigm is widely considered as an important class of truly distributed Grid applications which poses many challenges for a Grid computing environment. Still to this time, rather few real-world applications have been successfully ported as Grid-enabled workflows. We present the Askalon programming and computing environment for the Grid which comprises a high-level abstract workflow language and a sophisticated service-oriented runtime environment including meta-scheduling, performance monitoring and analysis, and resource brokerage services. We demonstrate the development of a real-world river modelling distributed workflow system in the Askalon environment that harnesses the computational power of multiple Grid sites to optimise the overall execution time.

1 Introduction

Grid computing simplifies the sharing and aggregation of distributed heterogeneous hardware and software resources through seamless, dependable, and pervasive access. It is well known that highly dynamic Grid infrastructures severely hamper the composition and execution of distributed applications that form complex workflows.

We have developed the Askalon programming and computing environment [5] whose goal is to simplify the development of Grid applications. Askalon currently supports the performance-oriented development of single-site, parameter study, and workflow Grid applications. The focus of this paper is on distributed workflow applications.

Workflow applications are first specified using a novel high-level abstract workflow language that shields the user from any Grid middleware implementation or technology details. The abstract representation is then mapped to a concrete workflow that can be scheduled, deployed, and executed on multiple Grid sites. The Askalon computing environment is based on a service-oriented

* This research is partially supported by the Austrian Grid project funded by the Austrian Federal Ministry for Education, Science and Culture under the contract GZ 4003/2-VI/4c/2004.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 454–463, 2005.

architecture comprising a variety of services including information service and resource brokerage, monitoring, performance prediction and analysis, reliable execution, and meta-scheduling. Although a variety of Grid programming systems exist, few concentrate on workflow applications, and even fewer are capable to support the development and execution of real-world workflow applications.

In this paper we describe the development of a workflow application in the Askalon Grid environment which extends an existing river modelling system that was previously developed to run on a sequential computer only. This work resulted in a real-world distributed workflow river modelling system that harnesses the computational power of multiple national Austrian Grid sites.

The next section describes the Askalon programming and computing environmentfor the Grid. Section 3 proposes several overheads that we use in the workflow performance analysis process. Section 4 shows the representation of a river modelling workflow application in the Askalon environment. We present in Section 5 a performance analysis study accompanied by a small overhead analysis that illustrates the benefit of executing the application in a distributed Grid environment. Section 6 concludes the paper.

2 Askalon Programming and Computing Environment

This section describes the Askalon programming and computing environment for the Grid, currently under development at the University of Innsbruck [5].

2.1 Workflow Specification Languages: AGWL and CGWL

In contrast to other approaches [1, 3, 6, 8, 10, 12–14], Askalon enables the description of workflow applications at a high level of abstraction that shields the user from the middleware complexity and the dynamic nature of the Grid. Although workflow applications have been extensively studied in areas like business process modelling and web services, it is relatively new in the Grid computing area.

Existing work on Grid workflow programming commonly suffers by one or several of the following drawbacks: control flow limitations (e.g., no loops), unscalable mechanisms for expressing large parallelism (e.g., no parallel sections or loops), restricted data flow mechanisms (e.g., limited to files), implementation specific (e.g., focus on Web services, Java classes, software components), and low level constructs (e.g., start/stop tasks, transfer data, queue task for execution) that should be part of the workflow execution engine.

Using the XML-based *Abstract Grid Workflow Language* (AGWL) [4], the user constructs a workflow application through the composition of atomic units of work called *activities* interconnected through control-flow and data-flow dependencies. In contrast to much existing work, AGWL is not bound to any implementation technology such as Web services. The control-flow dependencies include *sequences, Directed Acyclic Graphs*, *for*, *foreach*, and *while* loops, *if-then-else* and *switch* constructs, as also more advanced constructs such as

parallel activities (or master-slave patterns), *parallel loops*, and *collection iterators*. In order to modularise and reuse workflows, so called sub-workflows (or *activity types*) can be defined and invoked. Basic data-flow is specified by connecting input and output ports between activities. AGWL is free of low-level constructs as mentioned above.

Optionally, the user can link *constraints* and *properties* to activities and data flow dependencies that provide additional functional and non-functional information to the runtime system for optimisation and steering of the workflow execution on the Grid. Properties define additional information about activities or data links, such as computational and communication complexity, or semantic description of workflow activities. Constraints define additional requirements or contracts to be fulfilled by the runtime system that executes the workflow application, like the minimum memory necessary for an activity execution, or the minimum bandwidth required on a data flow link.

A transformation system parses and transforms the AGWL representation into a concrete workflow specified by the *Concrete Grid Workflow Language* (CGWL). In contrast to AGWL designed for the end-user, CGWL is oriented towards the runtime system by enriching the workflow representation with additional information useful to support effective scheduling and execution of the workflow. At this level, the activities are mapped to concrete implementation technologies such as Web services or legacy parallel applications. Moreover, a CGWL representation commonly assumes that every activity can be executed on a different Grid site. Thus, additional activities are inserted to pre-process and transfer I/O data and to invoke remote job submissions. Data transfer protocols are included as well. CGWL is also enriched with additional constraints and properties that provide execution requirements and hints, e.g., on which platform a specific activity implementation may run, an estimated number of floating point operations, or the approximate execution time of a given activity implementation.

During the compilation from AGWL to CGWL, several correctness checks are performed, like the uniqueness of names, the syntax of conditionals, or the existence of links. The data-flow loosely defined in AGWL is verified and completed, the data types are added to all the ports, and the compatibility of the data-links is validated. If possible, automatic data type conversions are added.

The CGWL representation of an AGWL specification serves as input to the Askalon middleware services, in particular to the Workflow Executor and the Meta-scheduler (see Figure 1).

2.2 Askalon Grid Services

Askalon supports the performance-oriented execution of workflows specified in CGWL through the provision of a broad set of services briefly outlined in this section. All the services are developed based on a low-level Grid infrastructure implemented by the Globus toolkit, which provides a uniform platform for secure job submission, file transfer, discovery, and resource monitoring.

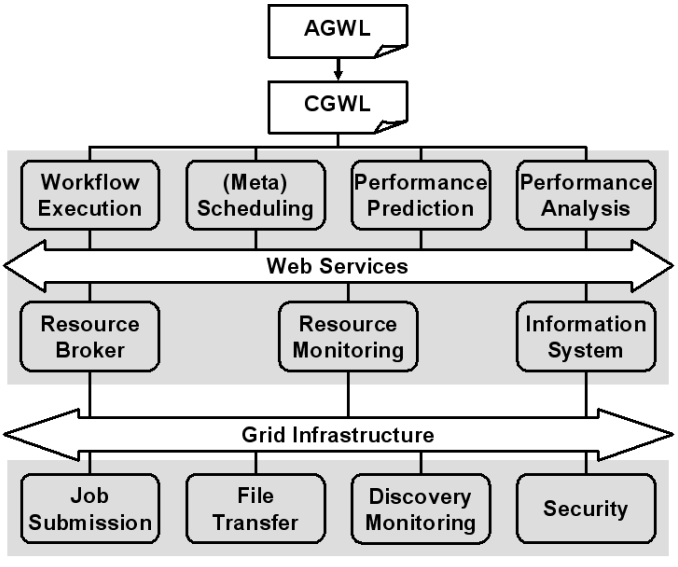


Fig. 1. The Askalon service-oriented architecture

Resource broker service targets negotiation and reservation of resources required to execute a Grid application [11].

Resource monitoring integrates and extends our present effort on developing the SCALEA-G performance analysis tool for the Grid [5].

Information service is a general purpose service for scalable discovery, organisation, and maintenance of resource and application-specific online and post-mortem data.

Workflow executor service targets dynamic deployment, coordinated activation, and fault tolerant completion of activities onto the remote Grid sites.

Performance prediction is a service through which we are currently investigating new techniques for accurate estimation of execution time of atomic activities and data transfers, as well as of Grid resource availability.

Performance analysis is a service that targets automatic instrumentation and bottleneck detection (e.g., excessive synchronisation and communication, load imbalance, inefficiency) within Grid workflows, based on the online data provided by the Monitoring service, or the offline data organised and managed by the Information service.

(Meta)-scheduler performs appropriate mapping of single or multiple workflow applications onto the Grid. We have taken a hybrid approach to scheduling single workflow applications based on the following two algorithms [9]:

1. *Static scheduling algorithm* approaches the workflow scheduling as an NP-complete optimisation problem. We have designed the algorithm as an instantiation of a generic *optimisation framework* developed within the ZENTURIO experiment management tool. The framework is completely generic

and customisable in two aspects: the definition of the *objective function* and the *heuristic-based search engine*. ZENTURIO gives first the user the opportunity to specify arbitrary parameter spaces through a generic directive-based language. In the particular case of the scheduling problem, the application parameters are the Grid machines where the workflows are to be scheduled. A heuristic-based search engine attempts to maximise a plug-and-play objective function defined over the set of generic annotated application parameters. For the scheduling problem, we have chosen the converse of the workflow *makespan* as the objective function to be maximised. For the implementation of the search engine we target problem-independent heuristics like gradient descent or evolutionary algorithms that can be instantiate the framework for other optimisation problems too (e.g., parameter optimisation, performance tuning). Our first search engine implementation is based on *genetic algorithms* that encodes the (arbitrary) application parameters (e.g., Grid machines for the scheduling problem) as *genes* and the parameter space as the complete set of *chromosomes*. We have conducted several experiments on real world-applications where a correctly tuned algorithm delivered in average 700% generational improvement and 25% precision by visiting a fraction of 10^5 search space points in 7 minutes on a $3GHz$ Pentium 4 processor.

2. *Dynamic scheduling algorithm* is based on the repeated invocation of the static scheduling algorithm at well-defined *scheduling events* whose frequency depends on the Grid resource load variation. The repeated static scheduling invocation attempts to adapt the highly-optimised workflow schedule to the dynamically changing Grid resources. Workflow activities have associated well-defined *performance contracts* that determine whether an activity should be migrated and rescheduled upon underlying resource perturbation or failure. We have conducted several experimental results in which our dynamic scheduling algorithm produced in average 30% faster execution times than the Condor DAGMan matchmaking mechanism [13].

3 Grid Workflow Overhead Analysis

In Askalon, a Grid workflow application is executed by the Workflow Executor service, based on the CGWL workflow representation. The workflow activities are mapped onto the processors available through the Grid using the (Meta)-Scheduler [9]. For each workflow activity A we currently compute three metrics:

1. *computation time* t_A of the corresponding remote Unix process, which we measure by submitting the job on the Grid as an argument to the POSIX-compliant "`/bin/time --portability`" program; the timing results are retrieved from the job's standard error stream;
2. *Grid execution time* T_A, measured between the events `STAGEIN` (i.e., when the input data is transferred to the execution site) and `COMPLETED` generated by the Globus Resource Allocation Manager used to submit the job on the Grid;

3. *Grid middleware overhead* associated with the activity A, which we define as: $O_A = T_A - t_A$.

With each workflow execution we associate a *Directed Acyclic Trace Graph* by unrolling the loops and cloning each activity upon every loop iteration. Since the number of parallel activities in our rather large workflow applications commonly exceed the available Grid machines, we introduce additional edges at the execution time which we call *run-time schedule dependencies*, that prohibit two parallel activities to execute on the same machine simultaneously because of the lack of additional Grid sites. For instance, if the parallel activities A_1 and A_2 are scheduled on the same Grid machine, an artificial run-time schedule dependency (A_1, A_2) is added to the set of workflow edges.

Let $(A_1, \ldots, A_n)$ represent a path in the trace graph of a workflow W which maximises the sum $\sum_{i=1}^{n} T_{A_i}$, also called the *critical workflow path*. We define the *Grid middleware overhead* of W as: $O_W = \sum_{i=1}^{n} O_{A_i}$.

Additionally, we measure the *communication overhead* generated by the (GridFTP-based) file transfers required by activities which are executed on different Grid sites having different NFS file systems.

4 River Modelling: Invmod

Invmod is a hydrological application for river modelling which has been designed for inverse modelling calibration of the WaSiM-ETH program [7]. It uses the Levenberg-Marquardt algorithm to minimise the least squares of the differences between the measured and the simulated runoff for a determined time period. Invmod has two levels of parallelism which are reflected in the Grid-enabled workflow version of the application depicted Figure 2:

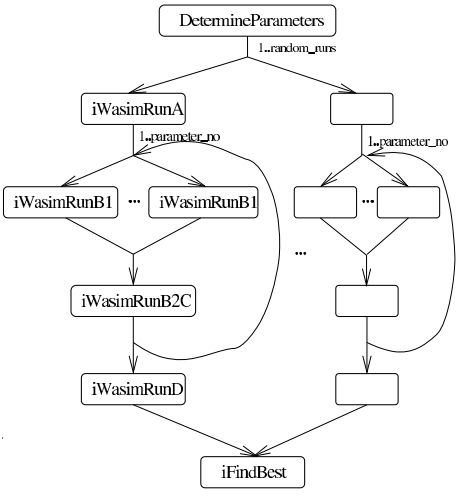


Fig. 2. The Invmod workflow

1. the calibration of parameters is calculated separately for each starting value using multiple, so called, *parallel random runs*;
2. for each optimisation step represented by an inner loop iteration, all the parameters are changed in parallel and the goal function is separately calculated.

The number of inner loop iterations is variable and depends on the actual convergence of the optimisation process, however, it is usually equal to the input maximum iteration number. Figure 3 represents an AGWL excerpt of the Invmod workflow, which contains the declarations of the internal `while` loop and `parallel-for` structures.

```
<agwl>
  <importATD url="http://.../invmodg/ATD/atd.xml" prefix="invmodgAtd"/>
  <workflow name="wfInvModG">
    <dataIn name="invModGIFile" source="wfInvModG/input.tar.gz"/>
    <body> ...
      <while name="while1">
        <dataIn name="repeatLoop" loopSource="wasimC/repeatLoop">
          <value>true</value>
        </dataIn>
        ...
        <condition>repeatLoop='true'</condition>
        <loopBody>
          <sequence name="seq3">
            <parallelFor name="pfor2">
              <dataIn name="noOfParams" source="a1/nrOfParameters"/>
              <loopCounter name="j" from="0" to="nrOfParams"/>
              <loopBody>
                <activity name="iWasimB1" type="invModGAtd:WasimB1">
                  ...
                  <dataIn name="wasimABData" source="w1/wasimABData"/>
                  <dataIn name="paramNr" source="pfor2/j"/>
                  <dataOut name="oFiles"/>
                </activity>
              </loopBody>
              <dataOut name="wasimB1OFiles" source="wasimB1/oFiles"/>
            </parallelFor>
            <activity name="iWasimB2C" type="invModGAtd:WasimB2C">
            ...
            </activity>
          </sequence>
        <loopBody>
        <dataOut name="loopResult" source="iWasimB2C/oFiles"/>
      </while>
    ...  </body>
    <dataOut name="result" source="FindBest/bestResult" saveTo="/tmp"/>
  </workflow>
<agwl>
```

Fig. 3. Excerpt from the Invmod AGWL representation

5 Experiments

The Askalon service-oriented architecture is currently being developed and deployed within the Austrian Grid infrastructure that aggregates several national Grid sites [2]. A subset of the computational resources which have been used for the experiments presented in this paper are summarised in Table 1.

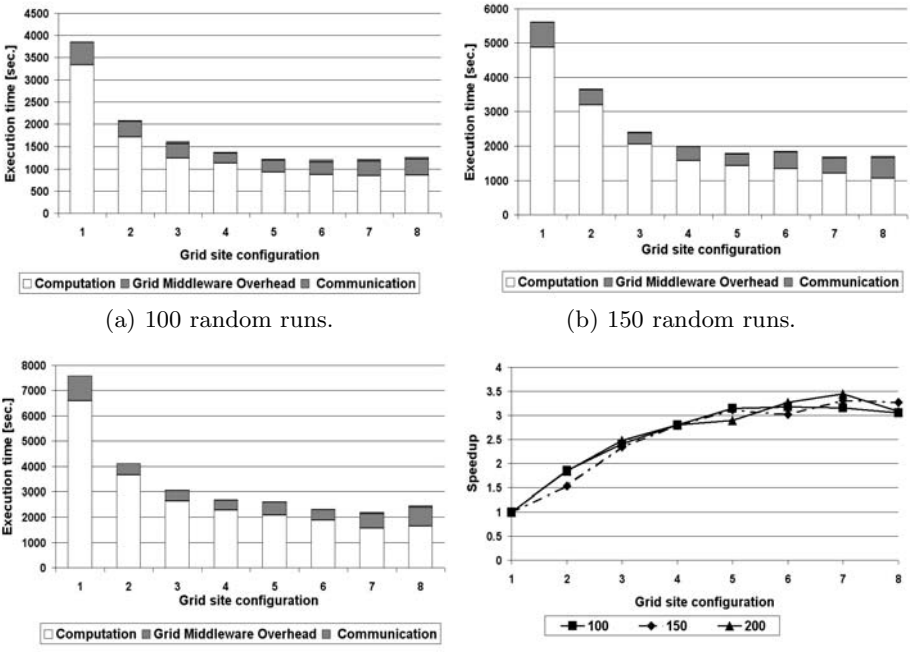
Table 1. The Austrian Grid testbed

Site	*Number of CPUs*	*CPU Type*	*Clock [GHz]*	*RAM [MBytes]*	*Location*
Hydra	16	AMD 2000	1.6	1000	Linz
ZID392	16	Pentium 4	1.8	512	Innsbruck
ZID421	16	Pentium 4	1.8	512	Innsbruck
ZID108	6	Pentium 3	1	256	Innsbruck
ZID119	6	Pentium 3	1	256	Innsbruck
ZID139	6	Pentium 3	1	256	Innsbruck
ZID145	6	Pentium 3	1	256	Innsbruck

The abstract AGWL representation of the Invmod workflow depicted in Figure 4 is translated into a concrete CGWL representation, which in which the activities are instantiated by legacy Fortran applications executed using the resource and data management support offered by the Globus toolkit. We performed three series of experiments for the Invmod river modelling application corresponding to three different problem sizes identified by 100, 150, respectively 200 parallel random runs. We first executed each problem size on the Hydra reference Grid site, since it is the fastest cluster in our Grid testbed for this application (faster than the Pentium 4). Then, we incrementally added new sites to the execution testbed to investigate whether we can improve the performance of the application by increasing the available computational Grid resources. For each individual execution, we measured the execution time as well as the overheads described in Section 3.

Figure 4 shows that the Invmod execution time improves by increasing the number of Grid sites. The best speedup is obtained when the two fastest clusters (i.e., Hydra and ZID392) are first joined to the resource pool. Less powerful clusters, however, also improve the overall execution but with a less steep increase in the speedup (see Figure 4(d)). As expected, the Grid middleware overhead increases by adding new slower clusters to the Grid testbed. This is mostly visible for the smallest problem size executed (i.e., 100 random runs), for which the large overhead/computation ratio produces the rather low increases in speedup (see Figure 4(a)). We obtained similar speedup curves for all the three problem sizes due to the limited number of Grid machines available. As the problem size gets larger, the ratio of the overhead to the overall execution time gets smaller and the speedup obtained are higher since the Grid machines perform more computation (see Figure 4(c)). Since the workflow schedules are computed by the meta-scheduler such that the data dependent activities are scheduled on the same sites (sharing the same NFS file system), the time spent on communication is negligible in all the experiments, even though the size of the generated files is of the order of Gigabytes.

The most important result is that by increasing the number of Grid sites, the overall performance of the distributed Invmod application improves compared to the fastest parallel computer available in the Grid infrastructure.



(a) 100 random runs.

(b) 150 random runs.

(c) 200 random runs.

(d) Speedup.

Fig. 4. The Invmod experimental results

6 Conclusions

In this paper we have shown the approach taken by the Askalon project for defining and executing Grid workflow applications. Askalon proposes a service oriented-architecture that comprises a variety of services for performance-oriented development of Grid applications, including resource brokerage, resource monitoring, information service, workflow execution, (meta-)scheduling, performance prediction, and performance analysis. Workflows are specified in a high-level abstract language that shields the application developer from the underlying Grid and its technologies. A transformation system instantiates the workflow into a concrete representation appropriate for Grid execution. We have demonstrated the effective use of Askalon for modelling, scheduling, executing, and analysing the performance of a real-world distributed river modelling application in the Austrian Grid environment. Our experiments based on workflow overhead analysis show that substantial performance improvement can be gained by increasing the number of sites available in the Grid environment (up to a reasonable number), compared to the fastest parallel computer available.

Currently we are applying Askalon to other real world applications from areas such as astrophysics and material science. Moreover, we are incrementally improving the Askalon middleware to better service the effective performance-oriented development of Grid applications.

References

1. Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Siebel Systems, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business process execution language for web services (bpel4ws). Specification version 1.1, Microsoft, BEA, and IBM, May 2003.
2. The Austrian Grid Consortium. http://www.austriangrid.at.
3. Dietmar W. Erwin and David F. Snelling. UNICORE: A Grid computing environment. *Lecture Notes in Computer Science*, 2150, 2001.
4. T. Fahringer, S. Pllana, and A. Villazon. A-GWL: Abstract Grid Workflow Language. In *International Conference on Computational Science. Programming Paradigms for Grids and Metacomputing Systems.*, Krakow, Poland, June 2004. Springer-Verlag.
5. Thomas Fahringer, Alexandru Jugravu, Sabri Pllana, Radu Prodan, Clovis Seragiotto Junior, and Hong-Linh Truong. ASKALON: A Tool Set for Cluster and Grid Computing. *Concurrency and Computation: Practice and Experience*, 17(2-4), 2005. http://dps.uibk.ac.at/askalon/.
6. IT Innovation. Workflow enactment engine, October 2002. http://www.it-innovation.soton.ac.uk/mygrid/workflow/.
7. K. Jasper. *Hydrological Modelling of Alpine River Catchments Using Output Variables from Atmospheric Models*. PhD thesis, ETH Zurich, 2001. Diss. ETH No. 14385.
8. Sriram Krishnan, Patrick Wagstrom, and Gregor von Laszewski. GSFL : A Workflow Framework for Grid Services. Technical Report, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, U.S.A., July 2002.
9. Radu Prodan and Thomas Fahringer. Dynamic Scheduling of Scientific Workflow Applications on the Grid using a Modular Optimisation Tool: A Case Study. In *20th Symposion of Applied Computing (SAC 2005)*, Santa Fe, New Mexico, USA, March 2005. ACM Press.
10. Ed Seidel, Gabrielle Allen, André Merzky, and Jarek Nabrzyski. Gridlab: a grid application toolkit and testbed. *Future Generation of Computer Systems*, 18(8):1143–1153, 2002.
11. Mumtaz Siddiqui and Thomas Fahringer. GridARM: Askalon's Grid Resource Management System. In *European Grid Conference (EGC 2005)*, Lecture Notes in Computer Science. Springer Verlag, February 2005.
12. Ian Taylor, Matthew Shields, Ian Wang, and Rana Rana. Triana applications within Grid computing and peer to peer environments. *Journal of Grid Computing*, 1(2):199–217, 2003.
13. The Condor Team. Dagman (directed acyclic graph manager). http://www.cs.wisc.edu/condor/dagman/.
14. Gregor von Laszewski, Beulah Alunkal, Kaizar Amin, Shawn Hampton, and Sandeep Nijsure. GridAnt-Client-side Workflow Management with Ant. Whitepaper, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, U.S.A., July 2002.

OpenMolGRID: Using Automated Workflows in GRID Computing Environment

Sulev Sild[1], Uko Maran[1], Mathilde Romberg[2], Bernd Schuller[2], and Emilio Benfenati[3]

[1] Department of Chemistry, University of Tartu, Tartu 51014, Estonia
{sulev.sild, uko.maran}@ut.ee
[2] Forschungszentrum Jülich GmbH, ZAM, D-52425 Jülich, Germany
{m.romberg, b.schuller}@fz-juelich.de
[3] Istituto di Ricerche Farmacologiche "Mario Negri" Via Eritrea 62, 20157 Milano, Italy
benfenati@marionegri.it

Abstract. Quantitative Structure Activity/Property Relationship (QSAR/QSPR) model development is a complex and time-consuming procedure involving data gathering and preparation. It plays an important role in the drug discovery pipeline, which still is mostly done manually. The current paper describes the automated workflow support of the OpenMolGRID system and provides a case study for the automation of the QSPR model development process in the Grid.

1 Introduction

The typical process for solving complex scientific problems involves the execution of time-consuming tasks that have to be carried out in a specific order. Often these interdependent tasks are carried out by independent applications that may require the manual processing of intermediate results by end-users in the middle of the process. While Grid computing provides a powerful infrastructure for making distributed computational resources available to compute intensive tasks, the automated workflows provide new ways to combine otherwise independent programs (or services) in the Grid environment to create a new form of applications designed specifically for the problem at hand. This article describes the automated workflow support of the OpenMolGRID system and provides a case study for the automation of the QSPR model development process.

1.1 Open Computing Grid for Molecular Science and Engineering

Open Computing Grid for Molecular Science and Engineering (OpenMolGRID) [1] is a project focused on the development of Grid enabled molecular design and engineering applications. *In silico* testing has become a crucial part in the molecular design process of new drugs, pesticides, biopolymers, and biomaterials. In a typical design process hundred thousands or even millions of candidate molecules are generated and their viability has to be tested. Economically it is not feasible to carry out an experimental testing on all possible candidates. Therefore, computational screening methods provide a cheap and cost effective alternative to reduce the number of candidates to a

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 464 – 473, 2005.

more manageable size. Over the years quantitative structure activity/property relationship (QSAR/QSPR) methods have been proved to be a reliable for the prediction of various physical, chemical and biological activities [2, 3].

The QSAR/QSPR methodology relies on a basic assumption that biological activity or physical property is a function of the molecular structure [4]. The molecular structure is characterized by theoretical parameters or so called molecular descriptors. Various statistical [5, 6] and variable selection [7] methods are then used to find quantitative relationships between experimentally available biological activity data and relevant molecular descriptors. The development of QSAR/QSPR models is rather complicated in practice, since various data pre-processing steps are required to prepare a proper training set before the model development can be started. The most common pre-processing tasks include the collection of experimental data, the generation of 3D coordinates for input structures, quantum chemical calculations, and molecular descriptor calculation. All these steps must be repeated in a proper sequence for each molecule in the training set. Traditionally, this kind of workflow involves a lot of manual labor and user interaction that is not practical when huge data sets are processed. The consistency of the data set is very important and the manual process may introduce unnecessary errors due to human factors. Computations in the data processing steps are time consuming, especially when quantum chemical calculations are involved.

The molecular design process can be significantly improved when Grid resources are exploited for the development and application of QSAR/QSPR models. Improvements are possible both for the reduced time necessary for the task of building a model (with all the above listed steps) and better quality resulting from the automation, which reduces mistakes and the variability of results. The OpenMolGRID project addresses the above-described problems by providing a Grid enabled infrastructure to automate these complex workflows and to speed up the process by distributing data parallel tasks over available computational resources.

2 Automated Workflows in Grid Environment

The specification and execution of complex processes like the process of molecular design and engineering using Grid resources is still an open field in Grid research and development. Solutions exist mostly for business processes. Languages to describe business processes are for example BPEL4WS (Business Process Execution Language for Web Services, see [8]) and WPDL (Workflow Process Definition Language, see [9]). The modeling of complex workflows in the scientific arena is mostly done manually using the tools some Grid middleware offers. The key point is the description of software resources available on Grid computing resources. These descriptions can be used for automated application identification and inclusion in multi-step workflows. The following sub-sections will describe the solution for automated workflow specification and processing developed within the OpenMolGRID project.

2.1 Workflow Specification

The primary question to be answered for identifying the necessary elements for a closed definition of a workflow is how applications and their interfaces are described in the environment at hand. In general workflows are built of tasks or processes as key elements. These elements are related through sequential temporal dependencies correlated to data flow between them or they are independent. OpenMolGRID uses the UNICORE Grid middleware [10] which offers workflow specification within a graphical user interface where tasks and sub-jobs are graphically linked to reflect dependencies and the necessary data flow is given through explicit transfer tasks. In addition workflow elements like loops, if-the-else, and hold are available to build up complex jobs. Applications or tasks within a job or sub-job are available on the client side as (application specific) plugins, which correspond to defined application resources on the server side. These resources are described by metadata that, among others, define the interface and I/O format information to clients.

Existing workflow description languages do not match the UNICORE model with respect to software resources. As these play the most important role within the automatic job generation a workflow specification language has been developed which allows a high level definition of various scientific processes containing sufficient information for the automatic generation of complete UNICORE jobs. This includes the tasks and their dependencies but also necessary resources for the steps. XML has been selected as a specification language for the workflow. A core element in a workflow is *task*, which has

- a *name* giving the identifier of a task fulfilled by an application resource and supported by a Client Plugin,
- an *identifier* giving the name for the UNICORE task in the job tree,
- an *id* giving the unique numerical identification within the workflow,
- an *export* flag specifying whether result files are to be exported to the user's workstation,
- a *split* flag specifying whether the task is data parallel and can be distributed onto several execution systems,
- a *splitterTask* giving the name of an application which is capable of splitting the input data for this task into n chunks,
- a *joinerTask* giving the name of an application which is capable of joining the n result files into one file, and
- *options* to feed the application with parameter settings.

For a *task* a set of simple resources can be specified requesting *runTime*, number of *nodes*, number of *processorsPerNode*, and *memoryPerNode*. For a *group* element of the workflow, which corresponds to a UNICORE sub-job the target system for the execution of all tasks within the group can be specified by *usite* and *vsite*. The workflow specification details are given in Appendix A.

Currently, there is no additional tool to generate the XML workflow; one has to use a standard text editor. With new UNICORE Client developments this will change.

2.2 Workflow Processing

A workflow specified as described above serves as input to the MetaPlugin, a special Plugin to the UNICORE Client. The MetaPlugin parses the XML workflow, creates a UNICORE job from it, and assigns target systems and resources to it. These tasks include a lot of sophisticated actions:

- Sub-jobs have to be introduced into the job wherever necessary, for example when requested applications are not available on the same target system;
- Transfer tasks have to be introduced into the job to ship data from one target system to another, which is target of a sub-job;
- Data conversion tasks have to be added between two tasks where the output format (specified in XML according to the application metadata) of one task does not match the input format of the successor task;
- Splitter and transfer tasks have to be added to the workflow as predecessor tasks of a splitable task for input data preparation;
- Sub-jobs have to be created around splitable tasks for each selected target system, and a transfer task to transfer the output data back to the superordnate sub-job;
- Joiner tasks have to be added to join the output data of split tasks;
- The directed acyclic graph of dependencies between all tasks (the explicit ones from the workflow specification and the automatically generated ones) has to be set up.

The MetaPlugin uses the resource information provided by the target system (*vsite*), the metadata of the applications, and information about the Plugins available in the Client. A so called resource information provider component has been developed to support the MetaPlugin in resource selection: It says which Client Plugin serves the task, which target system offers the application, and which are the I/O formats. Currently the MetaPlugin does resource selection at a very basic level but a more sophisticated resource broker component could easily be added.

The main advantage of this mechanism is that a user who wants to do model building can name the coarse-grained tasks and their dependencies in an XML workflow thereby avoiding the tedious job of the step-by-step preparation of the UNICORE job. The latter would afford detailed knowledge about for instance I/O formats for correct job preparation and the manual splitting and distribution of tasks onto appropriate target systems. Doing this automatically gives a lot of flexibility to the system to adapt to the actual Grid layout and resource availability and it helps avoiding human errors.

3 Case Study: Prediction of Solubility

The solubility is one of the most significant properties of chemicals and therefore important in various areas of human activity. Solubility in water is fundamental to environmental issues such as pollution, erosion, and mass transfer. Solubility in organic solvents forms much of the basis of the chemical industry. Solubility determines shelf life and cross contamination. Toxicity is critically dependent on solubility. Solubility is also linked to bioavailability and thus to the effectiveness of pharmaceuticals.

Solubility is one of the most important parameters of the ADME/Tox (absorption, distribution, metabolism, elimination and toxicity) profile that is used to test the drug ability (drug-likeness) of potential new drugs [11].

Therefore the computational prediction of solubility has been of huge interest, and the methods range from statistical and quantum mechanics to QSPR approaches [12]. The latter method is implemented in the OpenMolGRID system, an environment for solving the large-scale drug design and molecular design problems.

3.1 Description of the Solubility Data and Distributed Tasks

The example of using workflows in the Grid environment is given with the prediction of solubility in water. The example set of 178 data points consists of a large range of organic chemicals and is described in detail elsewhere [12].

The development of a QSPR model for the prediction of water solubility involves 4 distributed tasks in our Grid environment: (i) the 2D to 3D conversion of molecular structures; (ii) semi-empirical quantum chemical calculations of 3D structures; (iii) calculation of molecular descriptors for each 3D structure; and (iv) building up QSPR models. All those tasks are mapped onto geographically distributed resources and do need different amounts of computational resources, with the first and second step being the most demanding.

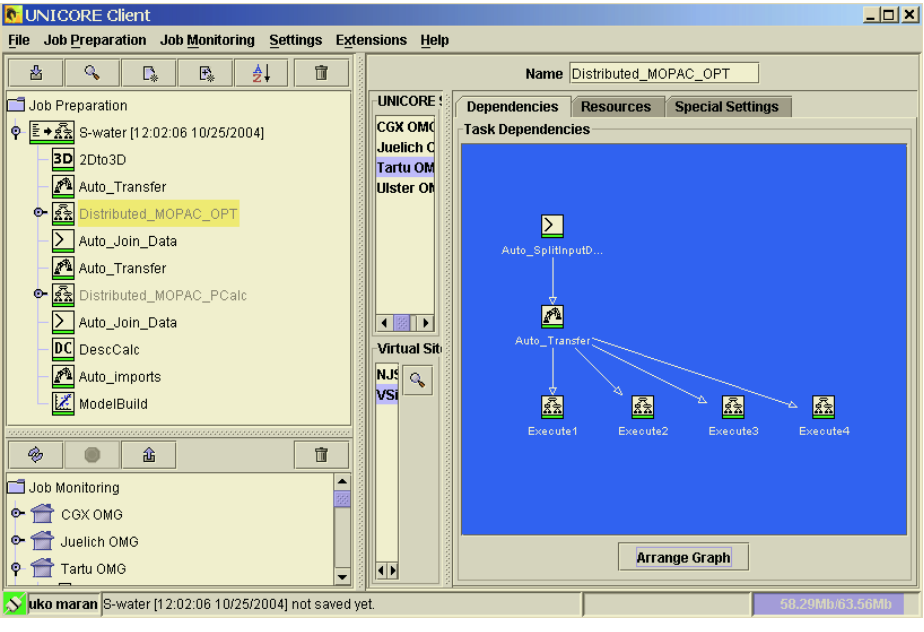


Fig. 1. Graphical representation of the OpenMolGRID workflow

The XML workflow used for the prediction of the solubility is given in Appendix B. As one can see the semi-empirical tasks are automatically split between available computational resources. Also different options can be set for the tasks. In the current

example a predefined set of keywords for the semi-empirical calculations is specified. It is not necessary to set them by hand when the workflow is reused. The graphical representation of the full workflow is given in the UNICORE Client Job Preparation area and the splitting of the semi-empirical task can be seen in the Task Dependencies area in Figure 1.

3.2 The QSPR Model for the Prediction of Water Solubility

The above described model development workflow has been carried out with the OpenMolGRID system and it produced a multi-linear QSPR equation (Table 1) with five descriptors for the prediction of the water solubility.

Table 1. The developed 5-descriptor QSPR model

Descriptors	Coefficient	*t*-test
Intercept	-0.81906	-5.48578
count of H-acceptor sites (MOPAC PC)	1.95510	22.60995
LUMO+1 energy	-0.27144	-7.77956
Min partial charge (Zefirov PC)	-13.80021	-13.18722
Number of rings	0.83094	8.03049
HA dep. HDSA-2/TMSA (MOPAC PC)	27.02062	7.32149

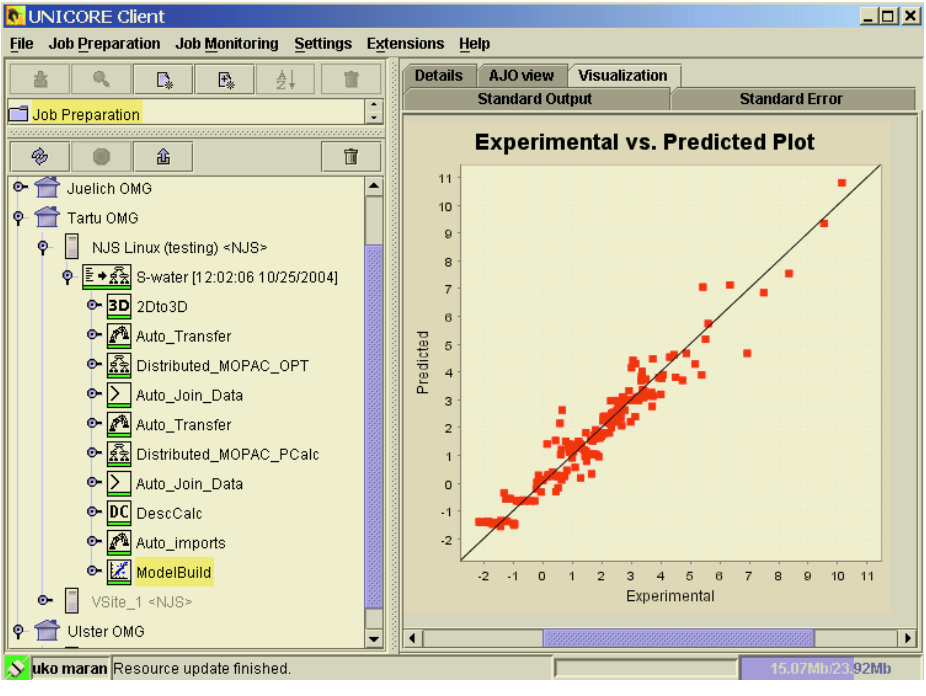


Fig. 2. Job Monitoring area with the finished workflow for Model Development and respective plot for 5-descriptor QSPR

The squared correlation coefficient, R^2, of this model equals to 0.94, its squared cross-validated correlation coefficient R^2_{CV} equals to 0.93, its F-value equals to 517.27, and its standard error of estimate, s, equals to 0.56. The R^2 and R^2_{CV} values are close to each other showing good predictive potential of the model. The analysis of the *t*-test values reveals that for the solubility the most important characteristics of the molecular structure are hydrogen bonding and minimum partial charge. Other characteristics of the structure are less important but influential like the number of aromatic and aliphatic rings in molecules. The plot of experimental versus predicted solubility values is given in Figure 2 together with the Job Monitoring area showing the successfully finished workflow.

4 Conclusions

The automated workflows in the Grid environment offer many attractive features to the end-users in many application domains, not limited to the molecular design. They reduce the time used for manual repetitive operations and allow convenient and transparent use of distributed resources. The automated workflows are user friendly and reduce the probability for human errors. In the above described model development process, the system located appropriate resources to carry out all the required tasks and automatically handled time consuming data conversion and transfer operations that normally involve manual processing. Workflows follow a unique defined procedure (once fixed) and thus it eliminates the problem of variability, related to the subjective choice of input parameters done by different users. For the above reasons, the results obtained with predefined workflows are easier to reproduce, a characteristic, which is very valuable in general and in particular for regulatory purposes (e.g. the assessment of toxicity by regulatory bodies).

Acknowledgements

Financial support is greatly acknowledged from the EU 5-th framework Information Society Technologies program (grant no. IST-2001-37238).

References

1. http://www.openmolgird.org
2. Katritzky, A. R., Maran, U., Lobanov, V. S., Karelson, M.: Structurally Diverse QSPR Correlations of Technologically Relevant Physical Properties. J. Chem. Inf. Comput. Sci.40 (2000) 1-18
3. Katritzky, A. R., Fara, D. C., Petrukhin. R., Tatham, D. B., Maran, U., Lomaka, A., Karelson, M.: The Present Utility and Future Potential for Medicinal Chemistry of QSAR/QSPR with Whole Molecule Descriptors. Curr. Top. Med. Chem. 2 (2002) 1333-1356
4. Karelson, M.: Molecular Descriptors in QSAR/QSPR. John Wiley & Sons, New York (2000).

5. Kowalski, B. R. (ed.): Chemometrics: Mathematics and Statistics in Chemistry. Nato Science Series: C:, Vol. 138. Kluwer Academic Publishers (1984)
6. Leardi R.: Nature-Inspired Methods in Chemometrics: Genetic Algorithms and Artificial Neural Networks. Elsevier Science (2003)
7. Maran, U., Sild, S.: QSAR Modeling of Mutagenicity on Non-congeneric Sets of Organic Compounds. In: Dubitzky, W., Azuaje, F. (eds.): Artificial Intelligence Methods and Tools for Systems Biology, Kluwer Academic Publishers, Boston Dordrecht London (2004) 19-36
8. Business Process Execution Language for Web Services. Version 1.0. 31-July-2002. (http://www-106.ibm.com/developerworks/library/ws-bpel1/) By Francisco Curbera (IBM), Yaron Goland (BEA Systems), Johannes Klein (Microsoft), Frank Leymann (IBM), Dieter Roller (IBM), Satish Thatte (Microsoft - Editor), and Sanjiva Weerawarana (IBM). Copyright 2001-2002 BEA Systems, International Business Machines Corporation, Microsoft Corporation, Inc.
9. zur Muehlen, Michael; Becker, Jörg: WPDL – State-of-the-Art and Directions of a Meta-Language for Workflow Processes. In: Bading, L. et al. (Ed.): Proceedings of the 1st KnowTech Forum, September 17th-19th 1999, Potsdam 1999
10. http://unicore.sourceforge.net/paper.html
11. Butina, D., Segall, M. D., Frankcombe, K.: Predicting ADME properties *in silico*: methods and models. Drug Discovery Today 7 (2002) S83-S88
12. Katritzky, A. R., Oliferenko, A. A., Oliferenko, P. V., Petrukhin, P., Tatham, D. B., Maran, U., Lomaka, A., Acree, W. E. Jr.: A General Treatment of Solubility. Part 1. The QSPR Correlation of Solvation Free Energies of Single Solutes in Series Solvents. J. Chem. Inf. Comput. Sci. 43 (2003) 1794-1805

Appendix A

XML Data Type Defintion for the specification of workflows:

```
<?xml version="1.0"?>
<!ELEMENT workflow ( ( (task*, group*) | (group*, task*) ),
                     dependency*, resourceRequest?)>
<!ELEMENT task (option*, localInput*, resourceRequest?)>
<!ELEMENT option EMPTY>
<!ATTLIST option
          name CDATA #REQUIRED
          value CDATA #REQUIRED
>
<!ELEMENT localInput EMPTY>
<!ATTLIST localInput
          source CDATA #REQUIRED
          destination CDATA #IMPLIED
          type CDATA #REQUIRED
          ascii (true|false) #IMPLIED
          overwrite (true|false) #IMPLIED
>
<!ATTLIST task
          name CDATA #REQUIRED
```

```
          identifier CDATA #REQUIRED
          id CDATA #REQUIRED
          export (true | false) #IMPLIED
          split (true | false) #IMPLIED
          splitterTask CDATA #IMPLIED
          joinerTask CDATA #IMPLIED
>
<!ELEMENT group (option*, ((task*, group*) | (group*,
task*)), dependency*, resourceRequest?)>
<!ATTLIST group
          type (subjob | repeat | doN | if | then | else)
                                                  #REQUIRED
          identifier CDATA #REQUIRED
          id CDATA #REQUIRED
>
<!ELEMENT dependency EMPTY>
<!ATTLIST dependency
          pred CDATA #REQUIRED
          succ CDATA #REQUIRED
>
<!ELEMENT resourceRequest ANY>
```

Appendix B

The XML workflow used for the prediction of solubility

```
<?xml version="1.0"?>
<!-- Model development for Solubility in Water -->
<workflow
xmlns="http://www.openmolgrid.org/namespaces/2004/Workf
lowDescription"
xmlns:rd="http://www.openmolgrid.org/namespaces/2004/Si
mpleResources">
<task name="2Dto3Dconversion" identifier="2Dto3D"
id="1" export="false" split="false">
<option name="molgeo.algorithm" value="Distance geome-
try"/><option name="molgeo.tolerance" value="3"/>
</task>

<task name="SemiempiricalCalculation" identi-
fier="MOPAC_OPT" id="2" export="false" split="true"
splitterTask="SplitStructureList" joiner-
Task="JoinStructureLists">
<option name="keywords" value="AM1 NOINTER MMOK
GNORM=0.1 EF"/>
</task>

<task name="SemiempiricalCalculation" identi-
fier="MOPAC_PCalc" id="3" export="false" split="true"
splitterTask="SplitStructureList" joiner-
```

```
Task="JoinStructureLists">
<option name="keywords" value="AM1 VECTORS BONDS PI
POLAR PRECISE ENPART MMOK 1SCF"/>
</task>

<task name="DescriptorCalculation" identi-
fier="DescCalc" id="4" export="false" split="false">
</task>

<task name="ModelBuilding" identifier="ModelBuild"
id="5" export="false" split="false">
<localInput source="H:\Unicore\test\Solub-data-
water.plf" destination="SolubData"
type="http://www.openmolgrid.org/namespaces/PropertyFil
e"/>
</task>

<dependency pred="1" succ="2"/><!-- 2D-3D to MOP1 -->
<dependency pred="2" succ="3"/><!-- MOP1 to MOP2 -->
<dependency pred="3" succ="4"/><!-- MOP2 to DC -->
<dependency pred="4" succ="5"/><!-- DC to MB -->
</workflow>
```

Implementation of Replication Methods in the Grid Environment

Renata Słota[1], Darin Nikolow[1], Łukasz Skitał[2], and Jacek Kitowski[1,2]

[1] Institute of Computer Science, AGH-UST, al.Mickiewicza 30, Cracow, Poland
[2] Academic Computer Center CYFRONET AGH, ul.Nawojki 11, Cracow, Poland

Abstract. Data replication methods are often used in Grid environments for improving the access time and data protection level. Dynamic replication allows the system to automatically create and delete replicas in order to follow dynamically changing system parameters and user access patterns keeping the performance high and resource usage in reasonable limits. In this paper we present the replication aspects of the Virtual Storage System developed within one of the grid projects. We also present results from replication tests done in a real grid environment.

1 Introduction

The Grids make possible usage of widely distributed computational and storage resources for solving scientific problems with scale and performance unseen before. A common problem in the scope of widely distributed computing is the distance[1] between the storage location where the requested data is kept and the computational resource location where these data are going to be used. This is even more important with the growing size of the data set since the time necessary to transfer the data may overcome benefits of the higher performance of distributed computing. One of the methods dealing with this problem is replication of data.

Data replication is the process of making additional copies of the original data. Data replication has been successfully used in the traditional computing systems, like mirroring (RAID systems) or backup. Three types of replication can be distinguished in the distributed systems: general replication, caching and buffering. The main difference between these types is in the place where the decision about making a copy of data is made and in the way of data removal. The decision of making a general replica is made by the server, while the decision of making a cache copy is made locally by the client. In the case of caching some data is being removed when the cache is full and new data have to be put there. This is done according to some data purging policy (like LRU). In the case of general replication the replicas are removed because the server decides they are

[1] The distance can be real geographical distance or a distance in the sense of latency and bandwidth parameters of the network connections.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 474–484, 2005.

no longer useful in this location. In the case of buffering the data is always copied and always removed after being consumed.

The replication of data in the Grid environments is done for two important reasons: improving the access time (and thus the performance) and increasing the data protection level. Additional benefits of using replication methods is the smaller system load for the storage resources as well as the lower network usage. On the other side there are some challenges and penalties to be paid when using replication. The main problem is making a decision about a replica creation. Obviously the replication methods need more storage capacity as the number of replicas increases. Additionally, the overall network traffic increases because of replica copying in time of creation and copying of data for keeping all the replicas consistent. Finding an efficient method for keeping all the replicas up-to-date in the cases of data modification is also challenging. Another problem is the need for additional software functionality which takes the role of managing replicated data sets. Finally the problem of determining which replica is the best for a given data access request should also be mentioned.

There are two main replication method categories: static replication and dynamic replication. The static replication is usually done manually by the user or administrator and is based on static parameters, while the dynamic replication is done automatically by the appropriate replica management component based on dynamically changing system parameters or user behavior. It can be based, for instance, on the statistical data about the previous accessing to the files or it can be based on predictions about the user behavior in the future.

The rest of the paper is organized as follows: Section 2 presents state of the art. Section 3 introduces the Virtual Storage System developed as part of the SGIgrid [1] project and the replication techniques used in it. Section 4 presents the test results. Section 5 concludes the paper.

2 State of the Art

In the DataGrid project [2] the replication was based on the tier model and the data was mostly read-only originating from a center location - CERN. The replication was static with best replica selection based on data access cost estimation.

In order to make an optimal replica selection some essential information concerning the network and the storage elements have to be available. The first kind of information is the bandwidth of network connections and the performance of the storage elements. The second kind of information is measurement data representing statistical or current system parameters - the measured network bandwidth and load of the storage elements. The third kind of information is data about the estimated performance parameters of the network and storage elements.

The cost of data access (in terms of access time) can be divided into two parts: the access cost imposed by the storage element and the access cost introduced by the network. Appropriate access cost model is introduced in [3]. The storage

element access cost estimation subsystem used in this model has been developed within the CrossGrid project [4].

For storage elements based on Hierarchical Storage Management (HSM) systems the access time can vary a lot from milliseconds to minutes. This estimation is not a trivial task since many parameters influence the access time: system load, the data location - tape or disk, number and availability of drives, size of file, etc. In our previous work we proposed a gray-box data access estimation system for HSM systems [5]. The estimation is done by event driven simulation of the essential (from the access time point of view) components of real HSM system.

The dynamic replication is an interesting topic and seems worth studying.

Foster et al. in [6] evaluate, using simulations, different dynamic replication strategies for managing large data sets in a high performance data grid. Their research has been done within the GriPhyN project [7]. They define six different replication strategies and test them by generating three different kinds of access patterns. They show how the bandwidth saving and latency differ with these access patterns depending on the applied strategy. The best strategies according to this paper are Fast Spreading and Cascading with Caching, but in the same time these strategies are the most storage consuming ones.

Park et al. in [8] present a simulation based study of a dynamic replication method called BHR (Bandwidth Hierarchy Replication). The BHR strategy takes advantage from the network-level locality which does not always fit to the geographical one. They show that their strategy outperforms other strategies in the case when hierarchy of bandwidth appears in the Internet.

Szymanski et al. in [9] study the use of highly decentralized dynamic replication services for improving the access time and bandwidth consumption of the overall system. They have assumed a two-tier data grid. They have tested two replication strategies: replicating to level 1 intermediate nodes (closer to tier 0) and replicating to level 2 intermediate nodes (closer to clients). Their simulation results show that the performance gains increase with the size of the data.

In the SGIgrid project, managed by ACC Cyfronet-AGH, instead of doing simulation research an attempt to study automatic replication methods by implementing and testing them in a real grid environment is being made.

3 Data Replication - SGIGrid Example

The SGIgrid project [1] aims to design and implement broadband services for remote access to expensive laboratory equipment as well as services for remote data-visualization. This goal is achieved by using the Grid technologies. The data produced during a remote experiment are stored for further visualization processing. Since the data and the visualization service can be far from each other the problem of efficient access to these data arises. The data replication method has been chosen for optimizing the access to data in terms of access time. The replicas are created and removed automatically by the system in order to achieve overall lower access time for client applications, not exhausting the storage resources in terms of capacity and the network in terms of bandwidth.

The replication functionality is provided by a subsystem called Virtual Storage System (VSS) being developed as part of the SGIgrid infrastructure.

The VSS is a storage system integrating the storage facilities, including Hierarchical Storage Management (HSM) systems, being part of the SGIGrid into a common storage infrastructure. It is built on top of Data Management System (DMS) [10] developed at the supercomputing center PCSS. Among the other functionalities like ordering of files and faster access to large files on tapes, VSS enhances the DMS with automatic data replication functionalities.

3.1 System Architecture

The VSS software is written in the Java language and uses the web services technology. The communication between the components is based on the SOAP [11] protocol. The system architecture is shown in Fig. 1. The gray boxes represent the components which has been added to the DMS. These extensions are Replica Manager, Log Analyzer (LA), Data Container Extensions (DCE) and API allowing file access to VSS from within an application.

The system consists of the following components:

- **Data Broker** - interface the VSS to the client. The purpose of this component is authentication and authorization of the user.
- **Metadata Repository, Log Analyzer** - the repository is the core of the system. It stores data concerning the virtual filesystem and the location of data. The Log Analyzer is part of the Repository. Its purpose is analyzing the history of data access operation and providing the result data to the Replica Manager.

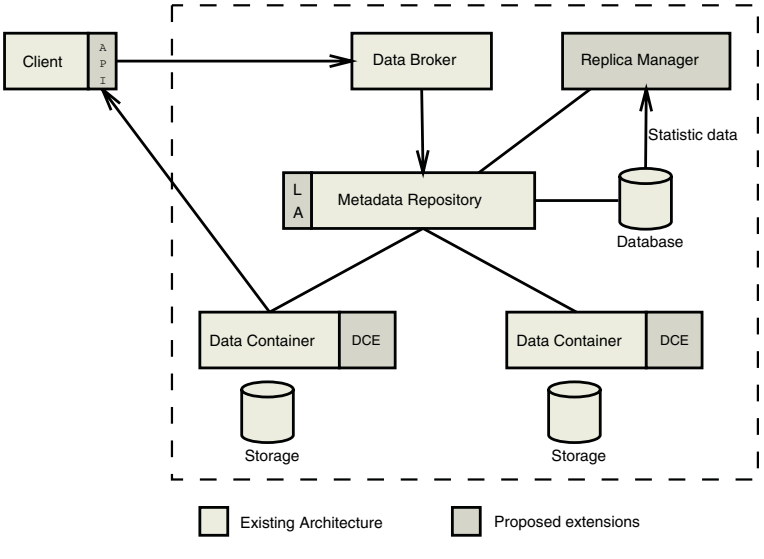


Fig. 1. Architecture of the VSS

- **Data Container, Data Container Extension** - Data Container is responsible for the physical data storage and retrieval. It has three parts:
  - **Managements Services** - provide services for data management to the Metadata Repository.
  - **DCE** - provides the new functionalities, i.e. file ordering, faster access to large files on tapes, data access time estimation.
  - **File transport** - provides file transfer services (servers for ftp, gsiftp, http).
- **Replica Manager** - the Replica Manger is responsible for the automatic creation and deletion of replicas.

The Replica Manager is described in more detail in the next subsection.

3.2 Replica Manager and Automatic Replication

The Replica Manager (RM) is responsible for the automatic creation and removal of replicas. The decision, made by the RM, when and where to create a replica is based on the statistic data provided by the LA.

The structure of the RM component is shown in Fig.2.

Replica Creation. The automatic replication is based on statistic data concerning previous data accesses. The Metadata Repository logs data access requests among the other events. The statistic data are gathered from this log by the LA in an XML file for easier later processing by the RM.

The following events are logged by the LA:

1. download of a file(GET),
2. upload of a file (PUT),
3. cancel upload of a file (ROLLBACK),
4. replica deletion (DEL).

Every event is described by the following record:

1. **action** - type of event (GET,PUT,ROLLBACK,DEL),
2. **location** - IP address of a client,
3. **fileID** - file identifier,

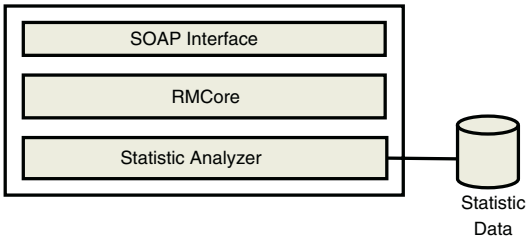


Fig. 2. Structure of the RM component

4. **container** - URL address describing data container and protocol,
5. **timestamp** - milliseconds since time zero (1970 January 01).

The algorithm for automatic replication, provided by the RM, takes as input parameter the records containing information about the previous file operations and returns as result a list of files with their replica destinations.

At the moment implemented algorithm for automatic replication takes under consideration the *frequency* of file downloads per user. Details are presented below.

Every couple location, fileID - (loc, fid) is considered separately and if it fulfills initial conditions (e.g. file size, number of replicas) the weight w is computed:

$$w(loc, fid) = \sum_{a \in L_r(loc, fid)} f(t_a), \tag{1}$$

where:
$L_r(loc, fid)$ – set of GET events for the file fid performed from location loc,
t_a – timestamp of an event a,
$f(t_a)$ – function specifying the influence of events age. It is defined as the following *cut-off* function:

$$f(t_a) = \begin{cases} 1 & \text{if } t_a \geq t_c - t_{max} \\ 0 & \text{if } t_a < t_c - t_{max} \end{cases}, \tag{2}$$

where t_{max} is the maximal age of event taken into account and t_c is the current time.

If $w(loc, fid) > \tau_c$, where τ_c is a threshold value, the RM decides to create a new replica of file fid. A data container to hold the new replica is selected, based on the user location, by the container selection algorithm described in paragraph "Replica Selection, Container Selection". If the file exists on selected data container or there is not enough free space for the new replica none replication is done, otherwise the file is replicated to this data container.

Removal of Replicas. Automatic replica removal, provided by the RM, is also based on statistic data. The algorithm used here is very similar to that used for automatic replication.

The following initial conditions have to be fulfilled for the replica removal:

1. the replica has been created by the RM - replicas, which were created manually, can be removed only manually,
2. the age of replica is greater, than the time period considered.

For every replica r, which fulfills the above initial conditions, the weight w is computed:

$$w(r) = \sum_{a \in L_r(r)} f(t_a), \tag{3}$$

where:
$L_r(r)$ – set of GET events for replica r,
t_a, $f(t_a)$ – the same as in formulas 1,2.

If $w(r) < \tau_r$, where τ_r is a removal threshold value, the RM decides to remove replica r.

Replica Selection, Container Selection. Replica and container selection is made by Metadata Repository. Selection of optimal replica (for an user request) is based on the following values:

1. network latency,
2. storage performance (read/write tests),
3. file access time estimation (significant for HSM),

Network latency measurement is performed using traceroute method, which is more reliable than ICMP Ping Request.

For location loc and for each data container c storing requested file fid, following weight d is computed:

$$d(c, loc, fid) = n(c, loc) + r(c) + w(c) + eta(fid) \tag{4}$$

where:
$n(c, loc)$ – network latency between data container c and user location loc,
$r(c)$ – read performance test,
$w(c)$ – write performance test,
$eta(fid)$ – access time estimation for file fid.

The optimal container is the one with minimal d.

Selection of data container for new replica is based on the above replica selection algorithm but takes under consideration only the network latency and the storage performance.

Replica Consistency, Propagation. Replica consistency is assured by Metadata Repository while the replica propagation is performed by the RM.

In the case of replica update Metadata Repository puts all other replicas in inconsistent state. Only consistent replicas are available for download. Metadata Repository inform RM about replica update. The RM is responsible for replica propagation. For this purpose we define two sets: sources and destinations. Initially sources set contains the updated replica and destinations set contains all other (inconsistent) replicas. Algorithm used for replica propagation is the Spanning Tree algorithm.

4 Test Results

4.1 Testbed Configuration

The VSS during the tests consisted of data containers localized in various sites in Poland. Data containers stored the data on heterogeneous storage systems (like

HSM systems and hard disk systems) with different data access performance characteristics. The network bandwidth between the sites ranged between 10 Mb/s to 625 Mb/s.

4.2 Testing Procedure

Ten files having different sizes (10-1000MB) have been stored on the VSS. In order to test the data replication facilities of the system a script simulating user activity has been implemented. The script first takes list of files to be accessed and next starts posting read requests to the system according to the specified access pattern. The script logs the file identifiers and the access times for these files. Few parameters allow us to control the behavior of the script program:

- *No_requests* - the number of requests to be issued,
- α - the alpha value for the Zipf-like pattern. 0 turns the pattern into completely random, while 1 will represent classical Zipf distribution,
- *interval* - the average time interval between requests; each interval is randomly generated,
- *interval_bias* - the maximal deviation from the average interval for a given time interval; the value should be between 0 and *interval*.

The tests are done in two phases. First, the script is executed on the client hosts located in different sites, and second, when the automatic replication triggered by the script execution is finished, the script is executed again with the same conditions. Then the results from the two runs are compared.

The tests were done for three different access patterns (α values). The replicas for the test files were removed before each test run.

4.3 Test Results

In the conducted experiment the client requests have been simplified to a read-whole-file requests. By access time to a file we assume the time from the moment of issuing a request till the moment the file is completely transfered. Taking into account the client requests for a given file an average access time value for that file has been calculated. The access time to a file requested by a client is considered as an access time to the best replica for this client. The results have been divided into three groups depending on the file size. The test results are shown in Fig. 3, 4, 5.

We can see that for small file sizes the performance gain is none or even negative. This is because of the replica selection algorithm. It is time consuming because it checks the estimated time of arrival for each container. It takes about 5-10 s for each container. For the medium and large files sizes the gain from replication increases.

The important costs associated with replication like increased storage capacity usage and possible higher network load is balanced by: wakeup period of RM and reaching a certain level of temporal locality by client requests. Additionally the file size and number of replicas thresholds can also be used to limit the replica creations.

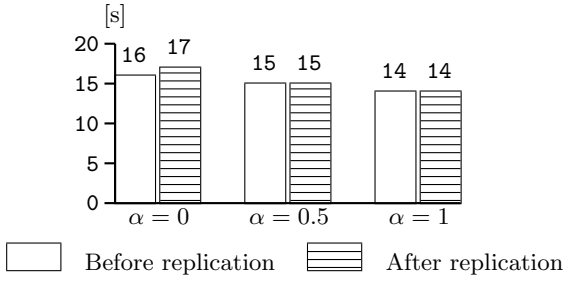


Fig. 3. Average Access Time - small files (10-20MB)

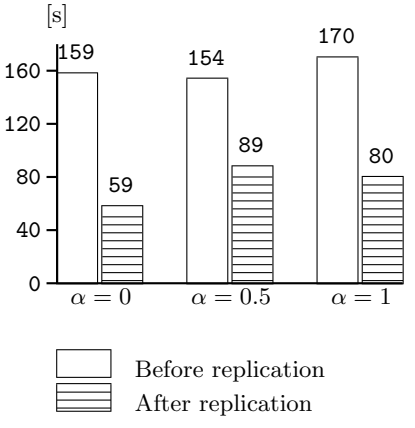


Fig. 4. Average Access Time - medium files (100-300MB)

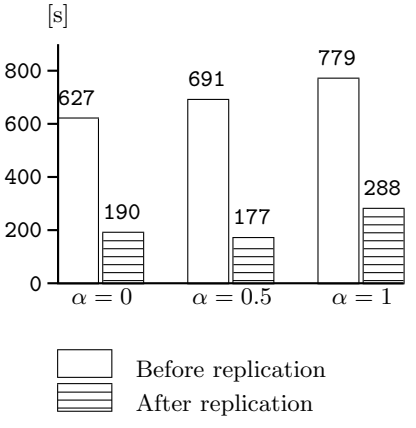


Fig. 5. Average Access Time - big files (512-1024MB)

5 Conclusions

In this paper we have presented an implementation of automatic data replication techniques for the Virtual Storage System developed within the SGIgrid project. The automatic data replication has been tested in a real grid environment. The tests have shown that the replication algorithm behaves properly and that the reduction of access time is more significant for larger files. For small files the replication didn't bring any performance gain, because of replica selection overhead. Achieved performance gain heavily depends on network architecture and data container's performance for low network and storage load. For highly loaded system the performance will mostly depend on the load balancing and scalability achieved by using replication.

The presented replication method is a user driven replication which seems reasonable for our framework where a geographically local group of scientist can

work together using the same data sets. In this case the gain can increase. Our further long term experiments with real requests clients will aim at studying such cases.

The tests were intended to show if some gain would be obtained due to the automatic replication. The penalty of replication is the higher usage of storage space. The growth of storage usage in the VSS is not fast due to the appropriate algorithm, which does not create new replicas too quickly. Additionally, the replicas that are unused for a long time are removed by a garbage-collector-like algorithm. Another aspect, not covered in this study, is the problem of system performance when the dynamics of replica creation and removal is much higher then in our current approach. In this case a replica may be removed even if it is used but the new replica coming in place is expected to bring higher overall gain. This aspect should be addressed in systems requiring faster system response due to the changes of user data access patterns.

Acknowledgments

The work described in this paper was supported by the Polish Committee for Scientific Research (KBN) project "SGIgrid" 6 T11 0052 2002 C/05836 and by AGH grant.

Thanks go to our colleagues from PCSS and CKPŁ for cooperation.

References

1. SGIgrid: Large-scale computing and visualization for virtual laboratory using SGI cluster (in Polish), KBN Project, http://www.wcss.wroc.pl/pb/sgigrid/
2. "DataGrid – Research and Technological Development for an International Data Grid", EU Project IST-2000-25182.
3. Stockinger, K., Stokinger, H., Dutka, Ł., Słota, R., Nikolow, D., Kitowski, J., "Access Cost Estimation for Unified Grid Storage Systems", 4-th Int. Workshop on Grid Computing (Grid 2003), Phoenix, Arizona, Nov 17, 2003, IEEE Computer Society Press.
4. "CROSSGRID – Development of Grid Environment for Interactive Applications", EU Project IST-2001-32243.
5. Nikolow, D., Słota, R., Kitowski, J. "Gray Box Based Data Access Time Estimation for Tertiary Storage in Grid Environment", in: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (Eds.), Parallel Processing and Applied Mathematics. 5th International Conference, PPAM 2003, Częstochowa, Poland, September 2003, LNCS, no. 3019, Springer, 2004, pp. 182-188.
6. Ranganathan, K., Foster, I., "Identifying Dynamic Replication Strategies for a High-Performance Data Grid", Proceedings of the International Workshop on Grid Computing, Denver, November 2001.
7. Grid Physics Network `http://www.griphyn.org/`
8. Park, S., Kim, J., Ko, Y., Yoon, W., "Dynamic Data Grid Replication Strategy Based on Internet Hierarchy", Lecture Notes in Computer Science Publisher, Springer-Verlag Heidelberg Volume 3033, 2004, pp.838-846.

9. Lamehamedi, H., Szymanski, B., Deelman, E., "Data Replication Strategies in Grid Environments", 5th Int. Conf. on Algorithms and Architectures for Parallel Processing, ICA3PP2002, Beijing, China, October 2002, IEEE Computer Science Press, Los Alamitos, CA, 2002, pp. 378-383.
10. PROGRESS, http://progress.man.poznan.pl/.
11. WebServices - SOAP http://ws.apache.org/soap/

A Secure Wrapper for OGSA-DAI

David Power, Mark Slaymaker, Eugenia Politou, and Andrew Simpson

Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford OX1 3QD,United Kingdom

Abstract. OGSA-DAI is a reference implementation of the OGSA Data Access and Integration services that facilitates homogeneous access to a wide range of databases, with this access being independent of the underlying database schemas and database management systems. In this paper we propose a secure wrapper for OGSA-DAI to allow an existing OGSA-DAI installation to be secured against inappropriate actions. The wrapper utilises XACML.

1 Introduction

OGSA-DAI [1] is a reference implementation of the OGSA [2] Data Access and Integration services. OGSA-DAI provides grid services that allow homogeneous access to a wide range of relational and XML databases, with this access being independent of the underlying database schemas or database management systems. As well as supporting simple query and update commands, it also allows data to be transferred to and from remote sites, the transformation of data, and the chaining together of actions to perform more complex tasks.

The current provision of support for security in OGSA-DAI is rather limited. The standard distribution of OGSA-DAI utilises a file that maps the distinguished name from a user's proxy certificate to a database username and password; all other security mechanisms have to be configured on the target database. Furthermore, there is—at the time of writing—no mechanism to restrict the use of other activities such as accessing remote sites or reading local files at the granularity of individual users. This approach would appear to be inconsistent with the widely held view that security is a *systems issue*—where consideration of the whole system is essential in order to reason appropriately about security properties. More positively, the OGSA-DAI mechanisms have been written in a modular fashion, which makes attempts to plug in additional security models relatively straightforward.

In this paper we propose a secure wrapper for OGSA-DAI that allows an existing OGSA-DAI installation to be secured against inappropriate use. We argue that shifting the configuration of users' rights away from the database has benefits. First, such an approach facilitates systems thinking, which is essential with respect to non-functional properties such as security. Second, it becomes possible to configure heterogeneous databases using a single unified policy—which, for some scenarios, is desirable. (This does not, of course, preclude individual sites

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 485–494, 2005.

opting-out of such a global policy and enforcing a policy that is consistent with local requirements.) Finally, it is then also possible to restrict other activities such as undesirable file access and connecting to remote sites.

The motivation for this work comes from the e-DiaMoND project [3], the main aim of which is to develop a prototype for a Grid-enabled national database of mammograms that is sympathetic to the work practices employed within the United Kingdom's NHS Breast Screening Programme. The core e-DiaMoND system consists of middleware and a virtualised medical image store to support the concept of a data grid. The virtualised medical image store comprises physical databases, with each being owned and managed by a Breast Care Unit (BCU). The e-DiaMoND grid is formed by participating BCUs coming together as a virtual organisation and uniting their individual databases as a single logical resource. In addition, there are a number of stand-alone databases and applications. The design of the underlying database has been described in a previous paper [4], as have the security issues surrounding the platform [5]. This paper brings these two key issues together.

The fundamental requirements for an access control model for a system such as e-DiaMoND can be stated simply: it should be flexible and it should be fine-grained. This simple statement of requirements is derived from the fact that there is no way of stating *a priori* what access control *policies* will be implemented at each site: some access control requirements will be nationally (or even internationally in the case of the European Union) mandated; others will be due to local policies and requirements; others to the quirks of departmental administrators. The best that one can do in such circumstances is to offer a system that is sufficiently flexible to accommodate these different needs. Our goal is to offer a model of access control that could support, on the one hand, one logical database with one national DBA and, on the other hand, every patient record being associated with exactly one hospital, which, in turn, has exactly one DBA. These needs can, perhaps, be made more concrete by means of examples.

One might imagine a national repository of mammograms and related patient data being an invaluable resource for studies of cancer trends. Having sought appropriate approval, a hospital might permit a clinician from a different part of the country to run queries across a subset of its patient information. This permission might, perhaps, be given to a particular individual under certain conditions—perhaps only access to those mammograms associated with smokers over 70 who died between 1995 and 2000 will be granted. It is thus necessary to offer means both of expressing this policy and enforcing it in a fashion that does not impact significantly upon performance.

As a second example, consider a woman who lives in west London—where her hospital records are based—but travels daily to east London to work. It would be more appropriate and convenient for the woman to attend for routine screening in east London—during her lunch break—than it would in west London. Therefore, the west London hospital offering remote access to that woman's information to the east London hospital would offer benefits for that individual, and would be exactly the kind of benefit one would expect from a national database that

supported the breast screening process. Again, specific secure access without a significant impact upon performance is required.

As well as addressing the specific needs of the e-DiaMoND project, our secure wrapper for OGSA-DAI offers several more generic benefits. First, the appearance of the wrapped OGSA-DAI would appear to clients just like OGSA-DAI grid services; as such, applications would not need redesigning or redeveloping to be compliant. Second, and a benefit that is key for systems such as e-DiaMoND, is the fact that the wrapper provides an exhaustive log of all users' interactions with the database (although this is, to be fair, relatively straightforward to achieve). A third benefit is that the approach allows access control policies to be configured in a variety of ways: homogeneously, with one policy covering the whole federated database, or heterogeneously, with each site implementing its own, individual, policy. Not only can security policies be different, but they may be configured at a number of different levels: the activities a user can access can be restricted, access to specific stored procedures can be restricted, and even the SQL queries passed to the database can be restricted/modified.

2 OGSA-DAI

Fundamentally, OGSA-DAI [1] consists of a set of grid services that facilitates interaction with databases via a common grid interface. Logically, OGSA-DAI consists of three main components: DAI Service Group Registry (DAISGR), Grid Data Service Factory (GDSF), and Grid Data Service (GDS).

The general pattern of usage is as follows. A user or application queries the DAISGR to find an appropriate GDSF. The user will then request that the GDSF creates a GDS, which it will subsequently communicate with directly. If the user or application wishes the GDS to perform a task, the user must create a perform document and pass it to the GDS. The GDS will then perform the actions described. If the actions involve interaction with the underlying database, then the GDS will obtain a user name and password from the database roles file.

Perform documents are XML documents containing a number of elements, with each such element representing an action to be performed. In this section we discuss a number of actions which are of particular interest. The reader should refer to the Grid Data Service documentation [6] for a more detailed description.

There are a number of activities which allow interaction with a relational database using a JDBC connection. First, an *sqlQueryStatement* element contains an *expression* tag, which, in turn, contains the SQL that will be executed. This is intended for read-only statements; it should be noted, however, that not all DBMSs support read-only connections.

An *sqlUpdateStatement* element also contains an *expression* tag which contains the SQL to be executed. This differs from *sqlQueryStatement* in that it is assumed that the statement will modify the database.

An *sqlStoredProcedure* element contains a *storedProcedureName* tag containing the name of the stored procedure. As is the case for *sqlQueryStatement* and

sqlUpdateStatement, it can also contain *sqlParameter* tags which contain either embedded values or a reference to a data source.

The *relationalResourceManagement* activity can be used to create and drop databases. The name of the database to be created or dropped is embedded in either the *createDatabase* or *dropDatabase* tag respectively.

One powerful feature of OGSA-DAI is the ability to send and receive data from third parties: in this respect, both synchronous and asynchronous activities are supported.

The default delivery mechanism is a *deliverToResponse* activity. This can be used to deliver the output stream of an activity within the GDS-Response document. This is, however, only one of many options.

There are activities to send and receive data from: GridFTP servers, other OGSA-DAI services using the Grid Data Transport portType, and a generic URL. It is the delivery of data from a URL that is of particular interest, as it provides an illustration of the need for access control.

The *deliverFromURL* activity can be used to pull data from a URL. It supports http, https, file and ftp. The URL is contained in the *fromURL* tag. Similarly, the *deliverToURL* activity can be used to deliver output to a URL as specified in the *fromURL* tag. By combining these two activities, it is straightforward to write a perform document to copy a file from the server hosting the OGSA-DAI service and then transfer it using ftp to another server. In the following example, the file being copied is the database roles file which contains the user names and passwords used to connect to the database. It should be noted that executing this perform document will cause an error to be reported, but only after the file has been transferred.

```
 <gridDataServicePerform
xmlns="http://ogsadai.org.uk/namespaces/2003/07/gds/types">
  <documentation>This will copy DatabaseRoles.xml</documentation>
  <deliverFromURL name="GetDatabaseRoles">
    <fromURL>
    file:///tomcat/webapps/ogsa/WEB-INF/etc/DatabaseRoles.xml
    </fromURL><toLocal name="theXMLFile"/>
  </deliverFromURL>
  <deliverToURL name="SendDatabaseRoles">
    <fromLocal from="theXMLFile"/>
    <toURL>ftp://guest@myserver.mydomain.com/DatabaseRoles.xml</toURL>
  </deliverToURL>
</gridDataServicePerform>
```

3 XACML

XACML (eXtensible Access Control Markup Language) [7] is an OASIS standard that allows one to describe—in terms of XML—two languages for a particular application. The policy language is used to describe general access control re-

quirements. The policy language has standard extension points that allow one to define aspects such as new functions, data types, and logics to combine such entities. The request/response language allows one to construct a query to determine whether a particular action should be permitted. Every response contains an answer pertaining to whether the action should be permitted in terms of one of four possible values: permit, deny, indeterminate, or not applicable. The first two of these values are self-explanatory. The indeterminate value denotes the fact that a decision cannot be made due to a missing value or the occurrence of an error. The not applicable value denotes the fact that this particular question cannot be answered by this particular service.

Typically, an application or user will wish to perform some action on a particular data resource. In order for this action to occur, a request to perform it is made to the entity that *protects* that resource. This *policy enforcement point* (PEP) might be, for example, a file system or a web server. On the basis of several aspects—the requester's attributes, the resource that the requester wishes to perform its action on, what the action is, and so on—the PEP will form a request. This request is then sent to a *policy decision point* (PDP). The PDP analyses the request together with a set of policies that apply to the request, and determines the appropriate answer with respect to whether or not the request should be granted. This answer is returned to the PEP. The PEP may then either allow or deny access.

The physical and logical relationships that exists between the PEP and the PDP are not prescribed: they may sit within a single application on the same machine; they may sit within a single application across different machines; they may exist in different applications on the same machine; or they may have no physical or logical connection. What is of interest to us is the potential for describing and enforcing access control policies via XACML.

A request is sent to a PDP via a *request document*. It is the responsibility of the PDP to find a *policy* that applies to that request.

A policy can have any number of *rules*—it is rules that contain the core logic of an XACML policy. At the heart of most rules is a *condition*, which is a Boolean function. If the condition evaluates to true, then the rule's *effect*—which is the intended consequence of a satisfied rule (either 'permit' or 'deny')—is returned. In addition, a rule specifies a *target*, which defines: the set of *subjects* that want to access the resource, the *resource* that the subject wants to access, and the *action* that the subject wishes to undertake on the resource.

Within the request document exist values of specific attributes that must be compared with the corresponding policy document values in order to determine whether or not access should be allowed. Request attributes consist of the target and the *environment*, which is a set of attributes that are relevant to an authorisation decision. These attributes are independent of a particular subject, resource, or action.

The values of the request attributes are compared with those of the policy document so that a decision can be made with respect to access permission. In the situation where many different policies exist—as opposed to the case in

which a single policy document exists—a *policy set document* is defined as a combining set of many policies.

A policy or policy set may contain multiple policies or rules, each of which may evaluate to different access control decisions. In order for a final authorisation decision to be made, combining algorithms are used, with *policy combining algorithms* being used by policy sets and *rule combining algorithms* being used by policies. Each algorithm represents a different way of combining multiple decisions into a single authorisation decision.

When the PDP compares the attribute values contained in the request document with those contained in the policy or policy set document, a response document is generated. The response document includes an answer containing the authorisation decision. This result, together with an optional set of *obligations*, is returned to the PEP by the PDP. Obligations are sets of operations that must be performed by the PEP in conjunction with an authorization decision; an obligation may be associated with a positive or negative authorization decision.

4 A Secure Wrapper for OGSA-DAI

In this section we describe our method of implementing our secure wrapper for OGSA-DAI. To meet the aim of ensuring that the wrapper appears like just another OGSA-DAI service, we utilise the code already available from OGSA-DAI. However, we only use the portions of the code that provide the interface and the extraction of the perform document. We propose an extension to the general process involved in the usage of an OGSA-DAI grid: this is illustrated in Figures 1 and 2. The wrapper provides a Wrapper Grid Data Service Factory (WGDSF) and a Wrapper Grid Data Service (WGDS), with the WGDSF and WGDS effectively acting as a level of indirection. The usage of the wrapper can be summarised as follows.

A request that would normally go to a GDSF actually goes to the WGDSF, as illustrated by action 1 in Figure 1. This request is generally signed by the user certificate of the requester. The WGDSF then sends a request (action 2), signed by a wrapper certificate, to the GDSF. The GDSF creates a GDS in the normal

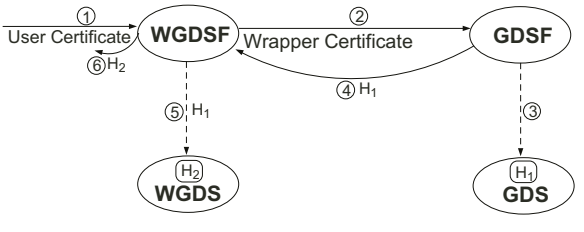


Fig. 1. Instantiation

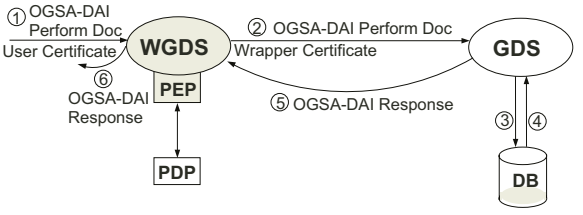


Fig. 2. Use

way (actions 3 and 4), and returns the handle to the WGDSF. The handle returned is used by the WGDSF as a parameter when it creates the WGDS (action 5). The handle to the WGDS is then returned to the requester (action 6). When the user or application subsequently wishes a task to be performed by the WGDS, they interact with it in exactly the same way as they would with a GDS. This interaction is shown in Figure 2, and can be described as follows.

An OGSA-DAI perform document signed by the user's certificate is passed to the WGDS (action 1 of Figure 2), as described in Section 2. This gives the impression to the user that they are interacting directly with an OGSA-DAI system.

Once the signed perform document is received by the WGDS several processes must be carried out to decide the correct response to the request. First, the perform document is extracted and then processed by the wrapper specific code. This will allow any changes to be made to the perform document, reflecting security restrictions, before forwarding it to a GDS instance.

The initial processing of the perform document involves extracting a list of activities that have been requested. This includes, amongst other items, the names of stored procedures along with any parameters, SQL statements and other activities. It is also necessary to check that the proxy certificate associated with the request is valid, i.e., that is it has not expired and it is signed by a trusted Certificate Authority (CA). If a proxy certificate has expired, or it is not signed by a trusted CA, then the request is rejected in exactly the same way as OGSA-DAI would do so.

Once the proxy certificate has been validated we need to extract the Distinguished Name (DN) from it. This is used as part of the XACML request as detailed in Section 5.

A logging system records various pieces of information, including: the DN, the contents of the perform document, the date and time of the request, the time before the proxy expires, and the source of the request (if known). The logging system could be a database or a simple log file: this is an implementation issue that can be decided on a case-by-case basis. It is, however, important that this logging system is secure and cannot be interfered with.

A request is then formulated so that it can be passed to the XACML PDP as described in Section 3 and indicated in Figure 2. This request contains all of the necessary information needed to make an informed decision about the

access requested. As a minimum, this must include the DN and details about the activities requested. If any parameters are passed in, then these also need to be forwarded to enable validation.

The PDP also needs to validate that not only does the DN have permission to execute a given activity with any parameters supplied, but also that this can be done at the current time.

The returned information will indicate the actual permissions that the system is prepared to grant to the user along with any obligations. If any of the activities requested are not authorised then the perform document will be rejected by returning an OGSA-DAI response document. This behaviour can be modified to allow the perform document to be changed so as it can be authorised. The amount of information that should be returned to the user when a rejection is given is open to debate. If the rejection is because of something like an expired proxy then this should be reported back. However, information pertaining to other errors may result in information flow, i.e., it may give away sensitive information pertaining to what is and is not stored in the database. It is however suggested that the method of detailing rejections should be consistent with OGSA-DAI.

The processing of SQL statements in a context such as this is a vast topic. The authors consider that full discussion of this is beyond the scope of this paper, and intend discussing this issue in a follow-on paper. We do, however, acknowledge that this is an important part of the overall security of a data access system. It is also worth noting that currently a perform document can have many activities that feed into each other. This can potentially cause problems unless the outputs and inputs to each activity are carefully analysed.

Any SQL statements need to be fully analysed to ensure that access restrictions imposed on users are not circumvented: attacks such as SQL injection are of particular relevance in this respect.

If the request document is valid, it is then passed to the target OGSA-DAI system. This is shown as action 2 in Figure 2. This request is passed to the GDS and is signed using a special system certificate that identifies the wrapper. The certificate used may be one of several and would be defined within the obligation returned from the PEP/PDP.

To increase security, a number of different certificates should be used, with these certificates corresponding to different users on the target database. A simple set of users could be a read-only user, a read/write user, and an admin user: this would help reduce the possibility of a read-only user being able to modify the database.

The modified perform document along with DN, date and time, etc. is again recorded to the logging 'database'. This provides a complete audit trail giving the history of the requests. It is also suggested that the OGSA-DAI system could sit on an intranet and thus only be available to requests originating from the 'OGSA-DAI Wrapper' system. Once the perform document is forwarded to the GDS the activities requested are carried out. The OGSA-DAI response document is returned to the WGDS (action 5 of Figure 2). The response document is logged

along with the DN, date and time, to the logging system. It is then returned to the user or application that made the initial request (action 6 of Figure 2).

5 Converting Perform Documents into Requests

Each activity in the perform document is converted into a separate XACML request, with the name of the activity being represented as the *action-id* in the request document using a string data type. The subject of each request is the distinguished name taken from the X509 certificate which was used to authenticate the user of the service. The distinguished name is stored as the *subject-id* in the request document using the XACML *X500Name* data type.

The values stored in the resources section of the request will depend on the type of activity which has been requested. For an *sqlQueryStatement* or an *sqlUpdateStatement*, the SQL statement stored in the *expression* tag is used as a *resource-id*, with any parameters being stored as separate *parameter* resources. If the values of the parameters are not known in advance then the implementation has two choices: either to execute each activity separately and insert the values into the request, or to only write policies which are applicable irrespective of the values of the parameters.

For an *sqlStoredProcedure* activity, the *resource-id* will be the name of the stored procedure, and any parameters will be stored as *parameter* resources as in the previous case.

For a *relationalResourceManagement* activity, the *resource-id* will be either *createDatabase* or *dropDatabase* depending on which tag is present. The name of the database will be placed in a *databaseName* resource.

For the *deliverToUrl* and *deliverFromUrl* activities the *resource-id* is either the *toURL* or the *fromURL*.

While it would have been possible to produce requests that represented the entire perform document, this would have required unnecessarily complex policy documents. A more effective long-term solution would be to add support to XACML for composition of requests, in the same way that policies can be composed using policy sets.

6 Discussion

We have described a secure wrapper for OGSA-DAI that utilises XACML with a view to allowing an existing OGSA-DAI installation to be secured against inappropriate actions. The work was driven by the needs of the e-DiaMoND project, and in addition to being of use to that project offers a number of more generic benefits that may be of relevance to the wider community. We would argue that—ultimately—the approach is very general and as such, it could be used to add an extra level of security to a wide range of grid services.

Some of the underlying technologies that the current technical realisation of our approach utilises will soon be surpassed; however we would argue that

the principles underpinning our approach and, indeed the wrapper itself, will remain largely unaffected by these changes. For example, from the GT3 toolkit, e-DiaMoND currently uses the Grid Security Infrastructure (GSI) to provide user authentication using X509 certificates; this will almost certainly be present in the forthcoming WSRF-based GT4. From OGSA-DAI we are using the request document, which is but one format in which a request could be encoded; any changes to this request document format would require a new mapping to an XACML request document, which compared to the intellectual effort expended would be a minor issue. The wrapper uses XACML version 1.1. Version 2.0 is at the time of writing (September 2004) under development, but it would be fair to assume that this later version will offer more, rather than less, functionality. This will hopefully allow the simplification of policies and request documents and increase the usefulness of this approach.

There are several potential areas of future work. Of particular relevance is the task of securing user activity permissions: how these are stored and accessed will be key to utilising this approach within a project such as e-DiaMoND. A more immediate pressing need—as discussed earlier—is the task of restricting and modifying SQL so that queries behave in accordance with global and local policies, and the potential for information flow is minimised.

References

1. Antonioletti, M., Jackson, M.: Product Overview. http://www.ogsadai.org.uk (2004) OGSA-DAI-USER-UG-PRODUCT-OVERVIEW-v5.0.
2. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: an open grid services architecture for distributed systems integration. www.globus.org/research/papers/ogsa.pdf (2002)
3. Brady, J.M., Gavaghan, D.J., Simpson, A.C., Mulet-Parada, M., Highnam, R.P.: eDiaMoND: A grid-enabled federated database of annotated mammograms. In Berman, F., Fox, G.C., Hey, A.J.G., eds.: Grid Computing: Making the Global Infrastructure a Reality. Wiley Series (2003) 923–943
4. Power, D.J., Politou, E., Slaymaker, M.A., Harris, S., Simpson, A.C.: An approach to the storage of DICOM files for grid-enabled medical imaging databases. In: Proceedings of the ACM Symposium on Applied Computing. (2004) 272–279
5. Slaymaker, M.A., Politou, E., Power, D.J., Lloyd, S., Simpson, A.C.: Security aspects of grid-enabled digital mammography. Methods of Information in Medicine (to appear) (2004)
6. Krause, A., Sugden, T., Borley, A.: Grid data service. http://www.ogsa-dai.org (2004) OGSA-DAI-USER-UG-GDS-v5.0.
7. Godik, S., Moses, T.: eXtensible Access Control Markup Language (XACML) version 1.1, committee specification. http://www.oasis-open.org (2003)

XDTM: The XML Data Type and Mapping for Specifying Datasets

Luc Moreau[1], Yong Zhao[3], Ian Foster[2,3],
Jens Voeckler[3], and Michael Wilde[2,3]

[1]Univ. of Southampton
[2] Argonne National Laboratory
[3] Univ. of Chicago

Abstract. We are concerned with the following problem: How do we allow a community of users to access and process diverse data stored in many different formats? Standard data formats and data access APIs can help but are not general solutions because of their assumption of homogeneity. We propose a new approach based on a separation of concerns between logical and physical structure. We use XML Schema as a type system for expressing the logical structure of datasets and define a separate notion of a mapping that combines declarative and procedural elements to describe physical representations. For example, a collection of environmental data might be mapped variously to a set of files, a relational database, or a spreadsheet but can look the same in all three cases to a user or program that accesses the data via its logical structure. This separation of concerns allows us to specify workflows that operate over complex datasets with, for example, selector constructs being used to select and initiate computations on sets of dataset elements—regardless of whether the sets in question are files in a directory, tables in a database, or columns in a spreadsheet. We present the XDTM design and also the results of application experiments with an XDTM prototype.

1 Introduction

In large open environments, users need to be able to access data stored in many different formats. An answer to this problem is the standardization of data formats and associated APIs. For example, XML is playing an increasingly important role in data exchange; FITS, the Flexible Image Transport System, is a standard method for storing astronomical data; and HDF5, the Hierarchical Data Format [5], is a file format (and associated software library) for storing large and complex scientific and engineering data.

Data and format standards are helpful in this context but are not general solutions in today's heterogeneous networked environments. Open environments must deal with legacy data, coexisting and evolving standards, and new data formats introduced to meet the needs of new applications or devices. Above all, open environments must be able to cope with evolution in data formats, so that ideally (for example) computational procedures do not require major changes to

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 495–505, 2005.

work with new formats. Format neutrality is hard to achieve, however, and we often encounter computational procedures that are no longer operational simply because their target data format no longer exists. The problem is that format dependency prevents code reuse over datasets that are *conceptually* identical, but *physically* represented differently.

BFD [11] and DFDL [2] offer partial solutions to this problem based on binary format descriptions: BDF offers converters from binary to some XML-based universal representation (and vice versa), whereas DFDL provides uniform access APIs to binary. They do not address the variety of existing datasets formats, however, since they focus on legacy binary formats. Moreover, they incur potentially expensive conversion cost to XML (both in time and space), and they do not promote reuse of computational procedures for logically equivalent datasets.

We propose a new approach to this problem based on a separation of concerns between logical and physical structure. We use a type system for expressing the logical structure of datasets, and we define a separate notion of a mapping to describe their physical representations. For example, a collection of environmental data might be mapped variously to a set of files, a relational database, or a spreadsheet but can look the same in all three cases to the user who accesses the data via its logical structure.

To explore the feasibility and applicability of these ideas, we define (and introduce here) XDTM, the *XML Dataset Type and Mapping* system. XDTM provides a two-level description of datasets: a *type system* that characterizes the abstract structure of datasets, complemented by a *physical representation description* that identifies how datasets are physically laid out in their storage. We adopt XML Schema [13, 3] as our type system; this technology has the benefit of supporting some powerful standardized query language that we use in selection methods. XDTM allows the specification of computational procedures capable of selecting subsets of datasets in a manner that is agnostic of physical representation.

Working within the GriPhyN project (www.griphyn.org), we have produced an implementation of XDTM and used it to manipulate datasets produced by the Sloan Digital Sky Survey (SDSS) [12]. Throughout this paper we use SDSS examples to demonstrate that the separation of concerns enabled by XDTM does indeed allow us to specify workflows that operate over complex datasets with, for example, selector constructs being used to select and initiate computations on sets of dataset elements—regardless of whether the sets in question are files, directories, or XML structures.

This paper is organized as follows. Section 2 reviews applications that use complex datasets and examines limitations that prevent their easy manipulation. Section 3 discusses the distinction between structure and format, while Section 4 describes how a type system, and specifically XML Schema, can be used to specify datasets structures. Section 5 focuses on a mapping that allows us to specify how a dataset's structure can be mapped to a physical representation. Section 6 overviews our prototype implementation. Section 7 discusses related work.

2 An Application Manipulating Complex Datasets

We introduce an application that has strong requirements for complex multi-format datasets. The Sloan Digital Sky Survey [12] maps one-quarter of the entire sky, determining the positions and absolute brightnesses of more than 100 million celestial objects, and measures the distances to more than a million galaxies and quasars. Data Release 2, DR2, is the second major data release and provides images, imaging catalogs, spectra, and redshifts for download.

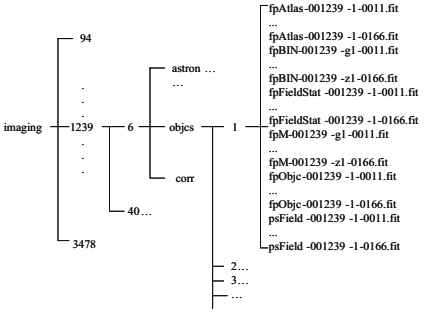


Fig. 1. Excerpt of DR2 (`http://das.sdss.org/DR2/data`)

We focus here on the directory `imaging` appearing at the root of the DR2 distribution, which we display in Figure 1. Inside it, we find a collection of directories, each containing information about a *Run*, which is a fraction of a strip (from pole to pole) observed in a single contiguous observing pass scan. Within a Run, we find *Reruns*, which are reprocessings of an imaging run (the underlying imaging data is the same, but the software version and calibration may have changed). Both Runs and Reruns directories are named by their number: in this paper, we will take Run 1239 and Rerun 6 as an example to focus the discussion. Within a Rerun directory, we find a series of directories, including `objcs`, *object catalogs* containing lists of celestial objects that have been cataloged by the survey. Within objcs, we find six subdirectories, each containing a *Camcol*, that is, the output of one camera column as part of a Run. In each Camcol, we find collections of files in the FITS format, the Flexible Image Transport System, a standard method of storing astronomical data. For instance, the *fpObjc* files are FITS binary tables containing catalogs of detected objects output by the frames pipeline. For these files, we see that the Run number (1239) and other information are directly encoded in the file name. From this brief analysis of a DR2 subset, we can see that DR2 is a complex hierarchical dataset, in which metadata is sometimes directly encoded in filenames, as illustrated for Run, ReRun, and CamCol directories and for all FITS files. Furthermore, DR2 is available in *multiple forms*: some institutions have DR2 available from their file system; it can also be accessed through the rsync and http protocols; alternatively, sometimes,

astronomers package up subsets directly as tar balls, such as specific Runs, for rapid exchange and experimentation.

3 Distinguishing Structure from Format

XML Schema, standardized by the World Wide Web Consortium [13, 3], provides a means for defining the structure, content, and semantics of XML documents. We assert in this paper that XML Schema can also be used to express the structure complex datasets. Increasingly, XML is being used as a way of *representing* different kinds of information that may be *stored* in diverse systems. This use of XML implies not that all information needs to be converted into XML but that the information can be viewed *as if* it were XML. Such an overall approach is also supported at the programming level. The Document Object Model (DOM) [9] is an application programming interface for well-formed XML documents: it defines the logical structure of documents and a programmatic way of accessing and manipulating a document.

Against this background, we decided to adopt a multilevel explicit representation of information and propose to make the *physical representation* (i.e., the format) of datasets an explicit notion, to coexist with the *abstract structure* of the dataset (i.e., its type). We strive to specify format separately from type, so that a dataset of a given type can be encoded into different physical representations. Programs that operate on datasets can then be designed in terms of dataset types, rather than formats, but can be executed regardless of the physical representations.

4 Type System for Describing Datasets Structure

We now introduce the type system that we use to specify the abstract structure of datasets. As in programming languages, we consider two forms of aggregations: *arrays* make their contents referenceable by indices, whereas *records* (or structures) make them referenceable by their name. As far as atomic types are concerned (i.e., types that cannot be decomposed into smaller elements), we consider the usual primitive types found in programming languages, such as int, float, string, and boolean. Furthermore, there exist numerous formats for encoding well-defined datasets, such as FITS files in astronomy or DICOM for medical imaging. We want such existing formats to be reusable directly, which means that such datasets should also be viewed as primitive types.

The type system must be language independent so that it can describe the structure of datasets independently of any programming or workflow language. All the features we discussed above (and more)—namely, array and recordlike aggregation, type naming, and attributes—are supported by XML Schema [13, 3]. We therefore propose to adopt the XML Schema, since it brings along other benefits by its standardized nature.

We show in Figure 2 an excerpt of the XML Schema for DR2. (For conciseness, we keep namespaces implicit in this paper.) The schema specifies that

```
<xs:schema targetNamespace="http://www.griphyn.org/SDSS"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.griphyn.org/SDSS">
  <xs:element name="imaging" type="Imaging"/>
  <xs:complexType name="Imaging">
    <xs:sequence>
      <xs:element name="run" type="Run" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Run">
    <xs:sequence>
      <xs:element name="rerun" type="ReRun" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="number" type="xs:nonNegativeInteger" use="required"/>
  </xs:complexType>
</xs:schema>
```

Fig. 2. XML Schema Excerpt for DR2

DR2 is composed of a single element named `imaging` of type `Imaging`. The complex type `Imaging` is a sequence of elements `run`, of type `Run`, with a variable (and possibly unbounded) number of occurrences. The complex type `Run` is a sequence of elements `rerun`, of type `Rerun`. Note that a mandatory attribute is being specified for type `Run`, with name `number` and natural value.

We note from this example that the type system provides a uniform mechanism for representing the structure of datasets both "within" and "outside" files. In fact, at the level of the type system, no construct distinguishes between what is inside a file and what is outside. Such a uniform approach allows us to express workflows and programs that operate both on datasets and file contents. Note that while XDTM makes it possible to express the structure of a file's content in the type system, this level of detail is not required. In particular, it may not be desirable to describe the structure of binary formats, and we may prefer to consider such types as opaque.

This use of XML Schema to describe the structure of datasets allows us to take the *conceptual* view that there is an XML document that represents a dataset encoded into a physical representation. Navigating a dataset can thus be expressed by using the conceptual XML view. We say that the XML document represents a conceptual view of a dataset because we do not expect every dataset to be translated into XML: to do so would defeat the purpose of what we are trying to achieve here, namely, to deal with complex data formats, as currently used in everyday practice.

Since XML Schema is standardized by the W3C, tools for editing and validating schemas exist; also, query languages such as XPath and XQuery are specifically conceived to operate on XML documents; moreover, the Document Object Model [9] offers a standard programmatic interface to such documents. We use these benefits in the rest of the paper, and we assume that the reader is familiar with the XPath notation [4]. We now turn to the problem of declaring the physical representation of datasets whose abstract structure has been specified by XML Schema.

5 Mapping to the Physical Representation of Datasets

We next present our approach to specifying the physical representation of datasets. First, since the structure of a dataset is specified by an XML Schema and

since a given dataset may have different physical representations, it is appropriate to characterize the physical representation in a document distinct from the schema, so that the schema can be shared by multiple representations. We expect the physical representation specification to refer to the Schema (using XML namespaces and conventions), but we do not want to annotate the schema directly with physical representation information (as does DFDL [2]), since this would prevent the schema from being reused for multiple representations of a same dataset.

Second, we recognize, like DFDL, the need to *contextualize* physical representations, so that we can allow a given type that occurs multiple times in a dataset to be encoded into different physical representations depending on its position in the dataset. Whereas annotation can directly be scoped by XML Schema's block structure, we now need to provide an alternative mechanism, since we have just precluded the use of annotations and therefore cannot take advantage of their natural scoping.

Third, our focus is on representing datasets, that is, aggregation of data, rather than on specifying the format of arbitrary (binary) data files (for example, for legacy reasons). Hence, we do not expect a single formalism for representing physical formats to be able to characterize arbitrary files in a descriptive manner. Instead, we support a mix of declarative and procedural representations; concretely, we adopt notations from which we can derive methods to read the physical representation of a dataset into an abstract one, and vice versa to write the abstract representation into a physical one. We anticipate that a library of such converters can be made available to the user and that new converters are permitted to be defined, hereby providing for extensibility. Such a pragmatic operational view of conversion to and from physical representation allows for direct reuse of legacy converters for pre-existing formats.

Requirements have identified that it is not desirable to add annotations to an XML Schema directly. Instead, we need a mechanism by which we can refer uniquely to an element in a dataset and indicate how it should be converted (remembering that XML Schema does not impose element names to be unique in a schema). Such a notation already exists: given that datasets structures are specified by XML Schema, we can use the XPath notation to identify the path that leads to a given element in a dataset.

We now introduce an XML notation for the mapping between abstract and concrete representation. Such a mapping is itself contained in a file so that it can be made explicit with datasets and published with their XML Schemas. XDTM (XML Dataset Type and Mapping) is the system that uses the schema and mapping documents in order to help users manipulate arbitrary datasets *as if* they were XML documents.

Figure 3 presents an excerpt of the mappings between abstract and physical representations for DR2. Each individual mapping identifies an element in DR2 using an XPath expression, and specifies its physical representation by one of the XML elements `directory`, `file`, `line`, `url`, and so forth. Specifically, Figure 3 states that the whole DR2 dataset is accessible from a URL; `imaging` and `run`

```
<xdtm:mappings>
    <xdtm:mapping path="/DR2">
        <xdtm:URL namingMethod="Self"  location="http://das.sdss.org/DR2/data"/>
    </xdtm:mapping>
    <xdtm:mapping path="/DR2/imaging">
        <xdtm:directory namingMethod="Self"/>
    </xdtm:mapping>
    <xdtm:mapping path="/DR2/imaging/run">
        <xdtm:directory namingMethod="RepresentRun"/>
    </xdtm:mapping>
    <xdtm:mapping path="//fpObjc">
        <xdtm:file namingMethod="RepresentFpObjc" type="opaque"/>
    </xdtm:mapping>
</xdtm:mappings>
```

Fig. 3. Excerpt of Mapping to/from Physical Representation

datasets are each represented by a directory; and `fpobjc` datasets are represented by files. Other mappings could refer to tar talls or zipped files.

For each kind of physical representation (directory, file, etc.), we use the attribute `namingMethod` to specify the name it is expected to have. The value of this attribute denotes a procedure that produces, from the abstract representation, a string that is the physical dataset's name (and vice versa). For example, `Self` returns the name of the element in the abstract representation; `RepresentRun` returns the value of the `run` attribute; `RepresentFpObjc` is used to create the fpObjc filename from field, run, and rerun attributes. In the case of fpObjc files, the mapping also specifies that the file type is "opaque"; that is, we do not describe the internal physical representation of this dataset.

6 Implementation

The aim of an XDTM implementation is to present, as DOM objects, datasets encoded in their physical formats so that they can be navigated by an XPath library like any other DOM object; symmetrically, it allows one to construct DOM objects that can be serialized to their physical representation. The implementation of XDTM consists of several components: *(i)* parsers for mappings and types, *(ii)* a library that reads datasets as DOM objects and vice versa, *(iii)* an XPath library that allows navigation of such DOM objects.

Our Java implementation uses the Jaxen XPath engine (jaxen.org), which processes XPath queries over DOM objects. Hence, the aim of our implementation is to materialize physical datasets as DOM objects so that they can be navigated by Jaxen, even though they refer to the actual physical representation of datasets. To this end, our implementation provides a library of classes implementing the standardized DOM interface [9] for all the physical representations found in DR2.

For each kind of physical representation, we have defined an associated class: directory in a file system (`VDSDirectoryElement`), reference to a http server through a URL (`VDSURLElement`), file containing an opaque or a fully described dataset (`VDSFileElement`), line within a file (`VDSLineElement`), and conjunction of multiple representations (`VDSGroupElement`). In addition, we need to allow for immediate values such as primitive types, which we abstract by the class

`VDSImmediateElement`. All these classes implement the DOM interface, which specifies standard methods to access and create node children, attributes, parent, and siblings. For example, the dataset `imaging` is mapped to a directory in Figure 3; in this case, XDTM internally uses the class `VDSDirectoryElement`, which implements accessor and creator methods for a directory in a file system.

In more detail, when the dataset `imaging` is read from the filesystem, it is represented by an instance of `VDSDirectoryElement` in the abstract representation. Whenever the dataset's children are accessed in the abstract representation, they are read from the contents of the directory in the filesystem and are made available as DOM elements themselves, all chained as siblings. The `namingMethod` attribute is also used at that point because the procedure it denotes reads the physical name and returns an element name in the abstract representation and potentially sets some attributes. Symmetrically, at the abstract level, one can instantiate the class `VDSDirectoryElement`, create new children for it, and insert it in other datasets. When saved into the physical representation, it is represented as a directory that contains its children encoded as files or directories. The `namingMethod` attribute value is again used to obtain the name of the directory from its abstract representation.

7 Related Work

The Data Format Description Language (DFDL) [2] is a descriptive approach by which one chooses an appropriate data representation for an application and then describes its format using DFDL. DFDL's ambitious aim is to describe *all* legacy data files, including complex binary formats and compression formats such as zip. This highlights a crucial difference with our approach: we seek not to describe binary formats but to express how data is aggregated into datasets. DFDL uses a subset of XML Schema to describe abstract data models, which are annotated by information specifying how the data model is represented physically. Specifically, mappings are declared as restrictions of primitive data types annotated by the concrete types they must be mapped to; the XML Schema is then decorated by scoped annotations specifying which mapping applies for the different types. Since the actual annotations are embedded inside an XML Schema, mappings and abstract types are not separable in practice. We see DFDL as complementary to our approach, however, since it can be used to convert XDTM atomic types.

The METS schema [10] is a standard for encoding descriptive, administrative and structural metadata regarding objects within a digital library. In particular, it contains a *structural map*, which outlines the hierarchical structure of digital library objects and links the elements of that structure to content files and metadata that pertains to each element. The METS standard represents a hierarchy of nested elements (e.g., a book, composed of chapters, composed of subchapters, themselves composed of text). Every node in the structural map hierarchy may be connected to content files that represent that node's portion of the whole document. As opposed to XDTM, METS does not separate an abstract data

structure from its mapping to physical representation, nor do physical aggregations such as complex directory structures or tar balls.

XSIL (XSIL: Java/XML for Scientific Data) comprises an XML format for scientific data objects and a corresponding Java object model, so that XML files can be read, transformed, and visualized [14]. Binary Format Description (BFD) [11] extends XSIL with conditionals; it can convert binary files into other binary formats; to this end, datasets are successively converted into XML by a parser that uses the BFD description of the input set, translated into another XML document using XSLT transformations, and finally converted by an "inverse parser" into another binary format, also specified in BFD.

Hierarchical Data Format 5, HDF5 [5], is a file format (and associated software library) for storing large and complex scientific and engineering data. HDF5 relies on a custom type system allowing arbitrary nestings of multidimensional arrays, compound structures similar to records and primitive data types. HDF5 introduces a *virtual file layer* that allows applications to specify particular file storage media such as network, memory, or remote file systems or to specify special-purpose I/O mechanisms such as parallel I/Os. The virtual file layer bears some similarity with our mapping, but focuses on run-time access to data rather than physical encoding. While HDF5 is not capable of describing legacy datasets that are not encoded as HDF5, some tools allow conversion to and from XML [8]. Specifically, the XML "Document Type Definition" (DTD) can be used to describe HDF5 files and their contents.

Microsoft ADO DataSet [1] is a "memory-resident representation of data that provides a consistent relational programming model regardless of the data source." Such DataSets can be read from XML documents or saved into XML documents; likewise, DataSet schemas can be converted into XML Schemas and vice versa. The approach bears some similarity with ours because it offers a uniform way of interacting with (in-memory copies of) datasets, with a specific focus on relational tables: however, the approach does not support interactions with arbitrary legacy formats. To some extent, the product XMLSpy (www.xmlspy.com) provides a symmetric facility, by allowing the conversion of relational data base to XML (and vice versa), but using XML Schema as the common model for interacting with data stored in different databases. It does not either provide support for datasets in non-relational formats.

8 Conclusion

We have presented the XML Dataset Typing and Mapping system, a new approach to characterizing legacy datasets such as those used in scientific applications. The fundamental idea of XDTM is to separate the description of a dataset structure from the description of its physical representation. We adopt the XML Schema type system to characterize a dataset's structure, and introduce a context-aware mapping of dataset types to physical representation. The separation of concerns between structure and physical representation allows one to define workflows and programs that are specified in terms of dataset struc-

tures, but can operate on multiple representations of them. This makes such computational procedures more adapted to deal with the evolution of data formats, typical of open environments. Beyond the SDSS application described in this paper, we have experimented with applications in in high energy physics and medical imaging, by which we demonstrate that the XDTM approach can help users to operate on concrete datasets used in many disciplines.

We are working on a robust implementation of XDTM capable of operating over large datasets, with checkpointing and restart capabilities. We are also designing the second generation of the GriPhyN Virtual Data Language, which will use XDTM to make workflows independent of the physical format of datasets they are intended to manipulate.

Acknowledgments

This research is funded in part by the Overseas Travel Scheme of the Royal Academy of Engineering, EPSRC project (reference GR/S75758/01), and by the National Science Foundation grant 86044 (GriPhyN).

References

1. Ado .net dataset. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcontheadonetdataset.asp.
2. M. Beckerle and M. Westhead. GGF DFDL Primer. Technical report, Global Grid Forum, 2004.
3. P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes. http://www.w3.org/TR/xmlschema-2/, May 2001.
4. J. Clark and S. DeRose. Xml path language (xpath) version 1.0. http://www.w3.org/TR/xpath, November 1999.
5. National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC). HDF5 Home Page. http://hdf.ncsa.uiuc.edu/HDF5/.
6. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.
7. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. The virtual data grid: a new model and architecture for data-intensive collaboration. In *Proceedings of the Conference on Innovative Data Systems Research*, Asilomar, CA, January 2003.
8. The HDF5 XML Information Page, jan 2004.
9. A. Le Hors, P. Le Hgaret, L. Wood, G. Nicol, J. Robie, Mike Champion, and Steve Byrne. Document object model (dom) level 3 core specification version 1.0. http://www.w3.org/TR/DOM-Level-3-Core/, April 2004.
10. Mets: An overview and tutorial. http://www.loc.gov/standards/mets/ METSOverview.v2.html, 2003.
11. J. Myers and A. Chappell. Binary format description (bfd) language. http://collaboratory.emsl.pnl.gov/sam/bfd/, 2000.

12. Sloan digital sky survey. `www.sdss.org/dr2`.
13. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. http://www.w3.org/TR/xmlschema-1/, May 2001.
14. R. Williams. Xsil: Java/xml for scientific data. Technical report, California Institute of Technology, 2000.

iGrid, a Novel Grid Information Service

Giovanni Aloisio, Massimo Cafaro, Italo Epicoco, Sandro Fiore, Daniele Lezzi, Maria Mirto, and Silvia Mocavero

Center for Advanced Computational Technologies,
University of Lecce/ISUFI, Italy
{giovanni.aloisio, massimo.cafaro, italo.epicoco, sandro.fiore, daniele.lezzi, maria.mirto, silvia.mocavero}@unile.it

Abstract. In this paper we describe iGrid, a novel Grid Information Service based on the relational model. iGrid is developed within the European GridLab project by the ISUFI Center for Advanced Computational Technologies (CACT) of the University of Lecce, Italy. Among iGrid requirements there are security, decentralized control, support for dynamic data and the possibility to handle user's and/or application supplied information, performance and scalability. The iGrid Information Service has been specified and carefully designed to meet these requirements.

1 Introduction

Flexible, secure and coordinated resource sharing among VOs requires the availability of an information rich environment to support resource discovery and decision making processes. Indeed, distributed computational resources, services and VOs can be thought of as sources and/or potential sinks of information. The data produced can be static or dynamic in nature, or even dynamic to some extent. Depending on the actual degree of dynamism, information is better handled by a Grid Information Service (static or quasi-static information) or by a Monitoring Service (highly dynamic information).

It is important to recall here the key role of information, and thus of Grid Information Services. High performance execution in grid environments relies on timely access to accurate and up-to-date information related to distributed resources and services: experience has shown that manual of default configuration hinders application performance. A so called grid-aware application can not even be designed and implemented if information about the execution environment is lacking. Indeed, an application can react to changes in its environment only if these changes are advertised. Self-adjusting, adaptive applications are natural consumers of information produced in grid environments.

However, making the relevant information available on-demand to applications is nontrivial. Care must be taken, since the information can be (i) diverse in scope, (ii) dynamic and (iii) distributed across one or more VOs. Information about structure and state of grid resources, services, networks etc. can be challenging to obtain. As an example, let us think about the problem of locating

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 506–515, 2005.

resources and/or services. Since grid environments are increasingly adopting architectures that are distributed and rather dynamic collections of resources and services, discovery of both resources and services becomes a difficult and complex task on large-scale grids. Thus, it is immediately evident the fundamental role of a Grid Information Service. It is, among many others, a key service that must be provided as part of a grid infrastructure/middleware.

The Globus Toolkit [2] version 2.x provides an LDAP based Information Service called Metadata Directory Service (MDS-2). This service has been deployed and used on many grid projects, both for production and development. However, the performances of the MDS Information Service were not satisfactory enough, and the Globus project provided users with a new version of this service, available in the Toolkit version 3.x and called Monitoring and Discovery Service (again, MDS-3) [9].

Version 3.x of the Globus Toolkit has been based on the Open Grid Service Infrastructure (OGSI) and the Open Grid Service Architecture [11] specifications. The MDS has been developed using Java and the Grid Services framework [10], but this service has not seen widespread adoption and deployment because it will change once again in the upcoming Globus Toolkit version 4. The new version, MDS-4, will be based on the emerging Web Service Resource Framework (WSRF) specification [12].

The GridLab project started adopting initially the Globus MDS, and some work has been done in order to extend it and to improve its performances [13]. Anyway, we decided to move from LDAP to the relational model, in order to overcome the Globus MDS shortcomings: a data model better suited for frequent reading, not able to cope with frequent updates, a weak query language missing key operations on data (e.g., there is no *join*), and performances. The relational data model allows us to model information and store data in tabular form, relationships among tables are possible, typical relational DBMS are strongly optimized for both fast reading and writing, and finally a good query language, namely SQL, allows complex operations on data, including joins.

The rest of the paper is organized as follows. We describe the iGrid Information Service in Section 2, and report iGrid performances in Section3. Finally, we conclude the paper in Section 4.

2 The iGrid Information Service

iGrid is a novel Grid Information Service developed within the European GridLab project [1] by the ISUFI Center for Advanced Computational Technologies (CACT) of the University of Lecce, Italy. Among iGrid requirements there are performance, scalability, security, decentralized control, support for dynamic data and the possibility to handle user's supplied information. The iGrid Information Service has been specified and carefully designed to meet these requirements. Core iGrid features include:

Web Service Interface built using the open source gSOAP Toolkit [5];

Distributed Architecture based on two kind of nodes, the iServe and the iStore Web services;

Based on Relational DBMS currently we are using PostgreSQL [6] as the relational back-end for both iServe and iStore nodes;

Support for Globus GSI through the GSI plug-in for gSOAP that we have developed;

Support for TLS (Transport Layer Security) Binding to DBMS to secure database communications;

Support for GridLab Authorization Service (GAS) to retrieve authorization decisions [7];

Support for Administrator defined Access Control List to allow for local decisions;

Support for Heterogeneous Relational DBMS through the GRelC library that we have developed;

Support for GridLab Mercury Logging Service to allow logging operations on the Mercury Monitor Service [8];

Support for Multiple Platforms iGrid builds on Linux, Mac Os X, Tru64, Aix and Irix;

Extensible by adding new information providers (this requires modifying the iGrid relational schema);

Extreme Performances as needed to support grid-aware applications;

Fault Tolerance see details in this section.

The iGrid distributed architecture, as shown in Fig. 1, is based on two kind of nodes, the iServe and the iStore GSI [3] enabled Web services. The iServe collects information related to a specific computational resource, while the iStore gathers information coming from registered iServes. The iGrid Information Service is based on a relational DBMS back-end and can handle information extracted from computational resources and also user's and/or application supplied information. iStores and iServes have the same relational schema for easy ingestion both on the local machine (iServe node) and remote machine (iStore node). The current architecture allows iStores to register themselves to other iStores, thus creating distributed hierarchies.

We recall here that the network topology closely resembles the one adopted by the Globus Toolkit MDS-2, however we use a completely different information dissemination strategy based on a push approach, whilst MDS-2 is based on a

pull approach. Even though the iGrid performances are alredy satisfactory (as shown later in Section 3), we are also investigating a possible implementation of a peer to peer overlay network based on one of the current state of the art distributed hash table algorithms in order to further improve iGrid scalability. The implementation includes system information providers outputting XML, while user's and/or application information is directly supplied by the user/application simply calling a web service registration method.

The information generated on iServe machines is stored locally on their PostgreSQL database using the GRelC library and binding securely with TLS to the database; moreover the information is also converted to XML using the GRelC MultiQuery protocol to preserve referential integrity among data belonging to different tables. Finally, it is sent to the iStore nodes.

The Web service interface is based on the gSOAP toolkit and on the GSI plug-in for gSOAP [4], a package we have developed to add GSI security to the iGrid Web Service. Our latest version of this software provides the following features:

- based on the GSS API for improved performances;
- extensive error reporting related to GSS functions used;
- debugging framework;
- support for both IPv4 and IPv6;
- support for development of both web services and clients;
- support for mutual authentication;
- support for authorization;
- support for delegation of credentials;
- support for connection caching.

We have designed iGrid taking into account security as follows. All of the connections from users/applications to iServes, from iServes to iStores and from iStores to hierarchical iStores are protected by means of a GSI channel, and all of the communications from an iServe or iStore to its local PostgreSQL database are protected by a TLS channel. This provides mutual authentication, confidentiality and anti-tampering, preventing snooping, spoofing, tampering, hijacking and capture replay network attacks. iGrid provides a fully customizable authorization mechanism, based on a GSI callback. Administrators are free to use the GridLab Authorization Service (GAS), the Globus Toolkit Community Authorization Service (CAS) or to implement their own authorization service. The iGrid daemon does not need to run as a privileged user, we sanitize the environment on iGrid startup and validate all user input. Special attention is devoted to the prevention of SQL injection attacks. For a typical SQL SELECT query the user's query string is escaped as needed and validated lexically and syntactically by an SQL parser that accepts only valid SELECT statements and rejects everything else.

Decentralized control in iGrid is achieved by means of distributed iServe and iStore services. Both creation and maintenance of system/user/application information are delegated to the sites where resources are actually located, and

administrators fully control how their site local policies are enforced. For instance, they decide about users/application that must be trusted for both data lookup and registration, if an iServe must register to one or more iStores, if an iStore must accept incoming data from remote iServes, the frequency of system information dissemination from iServes to iStores on a per information provider basis etc.

The implementation exploits the GrelC libraries [14], another software package we have developed targeting iGrid initially, and extended later to support the management of general purpose grid DBMS.The GRelC project supplies applications with a bag of services for data access and integration in a grid environment. In particular, iGrid relies on the following two libraries:

- Standard Database Access Interface (libgrelcSDAI-v2.0);
- MultiQuery (libgrelcMultiQuery-v1.0).

The former provides a set of primitives that allows the iGrid Information Service to transparently get access and interact with different data sources. This library offers an uniform access interface to several DBMSs exploiting a set of wrappers (that is, dynamic libraries providing dynamic binding to specific DBMSs) which can be easily plugged into our system. It is worth noting here that the SDAI library must take into account and solve the DBMS heterogeneity (different back-end errors, APIs and data types), hiding all of the differences by means of a uniform layer. To date, SDAI wrappers are developed for PostgreSQL (which represents the default DBMS used by the iGrid Information Service), MySQL and unixODBC data sources, providing interaction both with relational and not relational databases.

The MultiQuery library is built on top of the libgrelcSDAI and provides a set of APIs to develop an XML/SQL translator for the MultiQuery submission. The MultiQuery is a GRelC proprietary kind of query/format used to update a data source using a massive amount of data (this is a single shot query). The rationale behind this kind of query is that the user does not directly transfer the update query (INSERT, UPDATE and DELETE), but just the data in XML format from which the XML/SQL translator is able to infer the query itself. The MultiQuery format contains logical links which will be resolved to physical links by the XML/SQL translator on the DBMS side. Due to the nature, the mechanism and the performance of this query, the Multiquery is strongly recommended to populate relational Information Services. In the iGrid project the MultiQuery files are written by the Information Providers after gathering local data. Additional information related to a performance analysis in an European testbed of the MultiQuery can be found in [15].

The iGrid Information System can handle information related to:

System belongs to this class information like operating system, release version, machine architecture etc;

CPU for a CPU, static information like model, vendor, version, clock speed is extracted; but also dynamic information like idle time, nice time, user time, system time and load is provided;

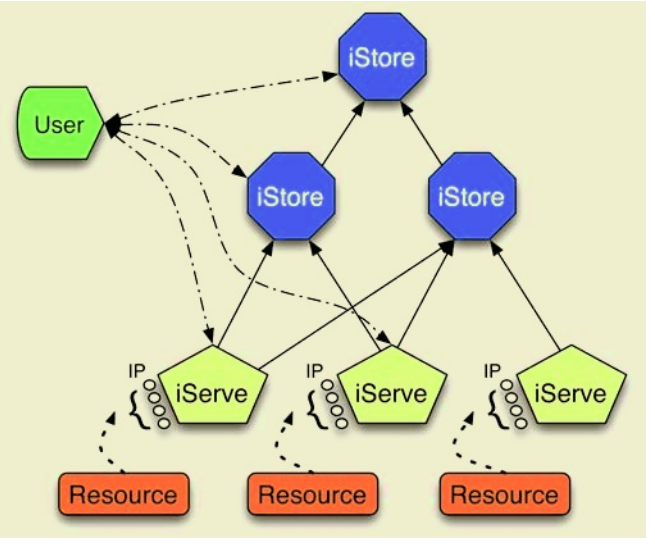


Fig. 1. iGrid hierarchical architecture

Memory related to memory, static information like RAM amount, swap space size is available. Dynamic information is related to available memory and swap space;

File Systems static as well dynamic information is extracted; some examples include file system type, mount point, access rights, size and available space;

Network Interfaces information that belongs to this category includes: network interface name, network address and network mask;

Local Resource Manager the information belonging to this category can be further classified as belonging to three different subclasses: information about queues, about jobs and static information about Local Resource Management System (LRMS). Some examples of information that can be extracted are: LRMS type and name; queue name and status, number of CPU assigned to the queue, maximum number of jobs that can be queued, number of queued jobs, etc; job name, identifier, owner, status, submission time etc. Currently information providers for OpenPBS and Globus Gatekepeer are available;

Certification Authorities information related to trusted Certification Authorities includes certificate subject name, serial number, expiration date, issuer, public key algorithm etc.

Information can be also supplied by users by invoking the appropriate Web service register methods. The user's supplied information includes:

Firewall name of the machine where the firewall is installed on, administrator name, the range of open ports allowed to pass through the firewall;

Virtual Organization information related to VOs can be used to automatically discover which resources belong to a given VO. In this category we have VO name, resource type, help desk phone number, help desk URL, job manager, etc;

Service and Web service this information can be used for Service or Web service discovery; information like service name, access url, owner, description, WSDL location, keyword is available.

The iGrid implementation relies on three main threads:

- *Web Service Interface* this thread is in charge of serving all of the SOAP requests coming from SOAP clients;
- *Collector* this thread handles the communications needed to publish iServe information into registered iStores. Periodically, the *Collector* will send to all of the registered iStores the information extracted by information providers, while user's and/or application supplied information is instead immediately sent to registered iStores. In case of failure of an iStore, iServes temporarily remove the faulty iStore from their registration list. Periodically, the iStore list is updated by adding previously removed iStores when iStores are available again. In this case, the local database is dumped and immediately sent to newly added iStores;
- *Information Provider Manager* this thread will activate one thread for each information provider listed in the iGrid configuration file. Upon information provider termination the extracted information will be immediately stored into the local database and copied in an appropriate spool directory for further pubblication on registered iStores by the *Collector* thread.

iGrid uses a push model for data exchange: as already explained, information extracted from resources is stored on the local PostgreSQL database, and periodically sent to registered iStores, while user's and/or application supplied information is immediately stored on the local database and sent to registered iStores. Thus, an iStore has always fresh, updated information on its local PostgreSQL back-end, and does not need to ask iServes for information. In contrast, the Globus Toolkit MDS-2 GIIS (Grid Index Information Service) needs to query remote GRIS (Grid Resource Information Service) servers when the information requested by the user is not available on the GIIS local cache. Indeed, MDS-2 uses a pull mechanism, and a cache expiry mechanism. This adversely affects MDS-2 performances.

The iGrid Information service does not need to exploit caching mechanisms in order to improve performances, since the underlying relational engine can answer complex queries efficiently, and all of the information is always available. Another mechanism we exploit to enhance performances is automatic stale information pruning. Each information is tagged with a time to live attribute that allows iGrid to safely remove stale information from the database as needed. Indeed,

on each user lookup data clean-up is performed before returning to the client the requested information. Moreover, when iGrid starts the entire database is cleaned up. Thus the user will never see stale information.

The iGrid Information Service provides fault tolerance: in case of failure of an iStore, iServes temporarily remove the faulty iStore from their registration list. Periodically, the iStore list is updated by adding previously removed iStores when iStores are available again. In this case, the local database is dumped and immediately sent to newly added iStores. Moreover, because of iGrid distributed and hierarchical architecture it is possible to implement a primary/backup approach by mirroring an iStore configuring all of the iServes to publish their information simultaneously on two or more different iStores.

The iGrid Information Service is used by the recently established SPACI consortium, that is building a grid among ISUFI/CACT, University of Naples, University of Calabria and Hewlett-Packard.

3 iGrid Performances

The performances of iGrid are extremely good. In what follows, we report the performance obtained for information retrieval on two testbeds. There were no high speed, dedicated networks connecting the machines belonging to the testbeds, and MDS2 servers were accessed without GSI authentication, while iGrid servers were accessed using GSI. The results were averaged over 50 runs.

Table 1 refers to tests related to the first testbed which comprises an iServe and a GRIS server hosted on one machine in Lecce, Italy, and an iStore and GIIS server hosted on another machine in Brno, Czech Republic. A query to both iStore and MDS2 GIIS was issued requesting all of the available information using clients in Lecce.

Table 2 refers to tests related to iStore/GIIS servers running on the second testbed which comprises four machines in Italy (three in Lecce and one in Bari, a city 200 Km far from Lecce) and one located in Poznan (Poland). The clients were in Poland, and the servers in Lecce. A total of four queries were issued. The first query in this set requested all of the information of each registered resource, the second one requested all of the information related to the machine in Poland, and the third one requested only the CPU information related to each registered resource. The last query was a complex query involving many filtering operations to retrieve information about attributes related to CPU, memory, network and filesystems.

Table 3 refers to tests related to iServe/GRIS servers running on the second testbed. Two queries were issued requesting respectively all of the information and information related only to the CPU using clients in Poland and servers in Bari.

As shown, iGrid performance is always much better than MDS2 when the cache is expired. When the cache is not expired iGrid performs like MDS2 and in some cases even better taking into account that iGrid uses GSI authentication while MDS2 do not on these testbeds. Finally, it is worth noting here that iGrid

Table 1. iStore / GIIS Performance on Testbed One

Search for information related to all resources	Cache expired (secs)	Cache not expired (secs)
GIIS	77	7
iStore	0.6	

Table 2. iStore / GIIS Performance on Testbed Two

iStore / GIIS		All res. info (secs)	Poland machine all info (secs)	All res. CPU info (secs)	Complex query (secs)
GIIS	cache expired	25,88	23,52	22,42	18.59
	cache not expired	2,53	1,64	0,46	0.67
iStore		3,06	1,01	0,92	0.67

Table 3. iServe / GRIS Performance on Testbed Two

iServe / GRIS		All info (secs)	CPU info (secs)
GRIS	cache expired	23,88	3,36
	cache not expired	1,05	0,39
iServe		0,95	0,81

answers very quickly even queries involving complex filtering on data. Since in grid environments it is difficult to always find an MDS2 server cache not expired, iGrid clearly represents a valid alternative to MDS2.

4 Conclusions

We have described iGrid, a novel Grid Information Service based on the relational model. The GridLab Information Service provides fast and secure access to both static and dynamic information through a GSI enabled Web service. Besides publishing system information, iGrid also allow publication of user's and/or application supplied information. The adoption of the relational model provides a flexible model for data, and the hierarchical distributed architecture provides scalability and fault tolerance. The software, which is open source, is freely available and can be downloaded from the GridLab project web site.

Acknowledgements

We gratefully acknowledge support of the European Commission 5th Framework program, grant IST-2001-32133, which is the primary source of funding for the GridLab project.

References

1. The GridLab project. http://www.gridlab.org
2. Foster, I., Kesselman C.: GLOBUS: a Metacomputing Infrastructure Toolkit, Int. J. Supercomputing Applications, 1997, pp. 115–28
3. Foster, I., Kesselmann, C., Tsudik G., Tuecke, S.: A security Architecture for Computational Grids. Proceedings of 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.
4. Aloisio, G., Cafaro, M., Lezzi, D., Van Engelen, R.A.: Secure Web Services with Globus GSI and gSOAP. Proceedings of Euro-Par 2003, 26th - 29th August 2003, Klagenfurt, Austria, Lecture Notes in Computer Science, Springer-Verlag, N. 2790, pp. 421-426, 2003
5. Van Engelen, R.A., Gallivan, K.A.: The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. Proceedings of IEEE CCGrid Conference, May 2002, Berlin, pp- 128–135
6. The PostgreSQL relational DBMS. http://www.postgresql.org
7. The GridLab Authorization Service. http://www.gridlab.org/WorkPackages/wp-6/index.html
8. Nmeth Zs., Gombs G., Balaton Z.: Performance Evaluation on Grids: Directions, Issues, and Open Problems. Proceedings of the Euromicro PDP 2004, A Coruna, Spain, IEEE Computer Society Press
9. Czajkowski K., Fitzgerald S., Foster I., Kesselman C.: Grid Information Services for Distributed Resource Sharing. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
10. Foster, I., Kesselmann, C., Nick, J., Tuecke, S.: Grid Services for Distributed System Integration. Computer, Vol. 35, 2002, No. 6, pp. 37–46
11. Foster, I., Kesselmann, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed System Integration. Technical Report for the Globus project. http://www.globus.org/research/papers/ogsa.pdf
12. The WSRF specification. http://www.globus.org/wsrf/specs/ws-wsrf.pdf
13. Aloisio G., Cafaro M., Epicoco I., Lezzi D., Mirto M., Mocavero S.: The Design and Implementation of the GridLab Information Service. Proceedings of The Second International Workshop on Grid and Cooperative Computing (GCC 2003), 7-10 December 2003, Shanghai (China), Lecture Notes in Computer Science, Springer-Verlag, N. 3032, pp. 131-138, 2004
14. Aloisio, G., Cafaro, M., Fiore, S., Mirto, M.: The GRelC Project: Towards GRID-DBMS, Proceedings of Parallel and Distributed Computing and Networks (PDCN) - IASTED, February 17 to 19, 2004, Innsbruck, Austria
15. Aloisio, G., Cafaro, M., Fiore, S., Mirto, M.: Early Experiences with the GRelC Library. Journal of Digital Information Management, Digital Information Research Foundation (DIRF) Press. Vol. 2, No. 2, June 2004, pp 54-60

A Grid-Enabled Digital Library System for Natural Disaster Metadata*

Wei Xing[1], Marios D. Dikaiakos[1], Hua Yang[2], Angelos Sphyris[3], and George Eftichidis[3]

[1] Department of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus
{xing, mdd}@ucy.ac.cy
[2] Department of Computer Science, Xian Institute of Post and Telecommunications, 710061 Shannxi, China
yanghua@xiyou.edu.cn
[3] Algosystems SA, 206 Syggrou Ave, 176 62, Kallithea (Athens), Greece
{asphyris, geftihid}@algosystems.gr

Abstract. The need to organize and publish metadata about European research results in the field of natural disasters has been met with the help of two innovative technologies: the Open Grid Service Architecture (OGSA) and the Resource Description Framework (RDF). OGSA provides a common platform for sharing distributed metadata securely. RDF facilitates the creation and exchange of metadata. In this paper, we present the design and implementation of a Grid-based digital library for natural-disaster research metadata. We describe the EU-MEDIN RDF schema that we propose to standardize the description of natural-disaster resources, and the gDisDL Grid service-based architecture for storing and querying of RDF metadata in a secure and distributed manner. Finally, we describe a prototype implementation of gDisDL using the Jena RDF Library by HP and the Globus 3 toolkit.

1 Introduction

European R&D projects and other related activities focusing on Natural Hazards and Disasters (earthquakes, floods, forest fires, industrial hazards, landslides, and volcano eruptions) produce results in the form of explicit or tacit knowledge represented by reports, project deliverables, data-sets derived from field work, interesting training and dissemination materials, etc. These artifacts are usually published and described in Web sites maintained by project partners during the duration of the respective projects. Following project completion, however, project teams dissolve and Web-site maintenance and support gradually fade out. Hence, general-purpose search engines are used to search for past-project results. Nevertheless, search-engine query results provide large numbers of unrelated links. Furthermore, hyperlinks pointing to potentially useful material do not come with references or additional links to adequate information describing

* Work supported in part by the European Commission under the EU-MEDIN project (grant agreement no. EVG1-CT-60003).

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 516–526, 2005.

the "value" of identified resources. Consequently, the identification of and access to useful information and knowledge becomes very difficult. Effectively, valuable knowledge is lost and it is practically impossible to find and take advantage of it.

To cope with this situation and to make research artifacts in the field of natural disasters widely and easily available to the research community, the European Commission has commissioned to Algosystems S.A. the development of a thematic Web portal to support the storage and retrieval of metadata pertaining to results of research in natural disasters [3]. Project-related metadata can be inserted via a Web interface to a back-end database. Interested researchers can use the "EU-MEDIN" portal to query the database and search for project-artifacts. This approach, however, is based on a platform-specific solution. A change in database technology or a need for upgrading the hardware platform or the operating system may dictate the transformation and storage of metadata in a different data model, format, and database system, hence incurring significant cost. Furthermore, the existence, update, and maintenance of the particular site is guaranteed only during the life-cycle of the project that undertook the portal's design, development, and maintenance. Also, the functionality of the site requires that end-users have to connect through the Web and register their metadata centrally; this makes it difficult, however, to extract all metadata related to a particular project and possibly submit them to a third party in batch. Last, but not least, there is a need to describe the metadata in a common and open format, which can become a widely accepted standard; the existence of a common standard will enable the storage of metadata in different platforms while supporting the capability of distributed queries across different metadata databases, the integration of metadata extracted from different sources, etc.

In this paper, we present **gDisDL**, a system designed to address some of the problems mentioned above. Our approach comprises:

- A schema for describing project-related metadata in a platform-independent form, using the *Resource Description Framework* (RDF). RDF is a general framework for describing metadata of Internet resources and for processing these metadata; it is a standard of World Wide Web Consortium (W3C). RDF supports the interoperability between applications that exchange machine-understandable information on the Web.
- A digital library system enabling the collection and storage of RDF-encoded metadata in distributed repositories, and the retrieval thereof from remote sites. This library is implemented as a Grid-service architecture comprising a set of Grid services, which allow the storage, management, and query of RDF metadata in a secure and distributed manner. To develop the library we use the Globus Toolkit 3 [18] for programming Grid services and the Jena toolkit [1] for handling RDF data.
- A set of graphical-user interfaces developed in Java to enable authorized end-users to create RDF metadata for natural-disaster research artifacts and to conduct keyword-based searches in RDF repositories.

The remaining of this paper is organized as follows. In section 2, we introduce Semantic Web technologies and Grid that used in our work and review related work. Section 3 describes matadata representation of the gDisDL system. We present the design challenges and architecture of the gDisDL system in section 4. Finally, we conclude our paper in Section 5.

2 Background

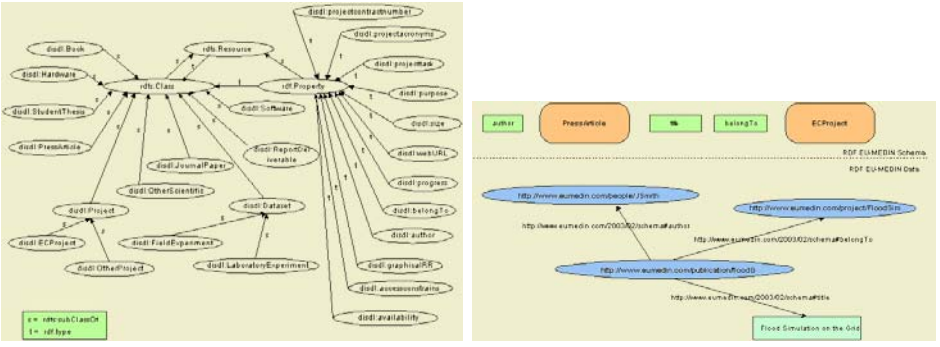
Resource Description Framework (RDF): The Resource Description Framework (RDF) is a language mainly used for representing information about resources on the World Wide Web [15]. In particular, it is intended for representing metadata about documents or other entities (e.g, Web resources, software, publications, reports, image files etc), such as the title, author, modification date, copyright, and licensing information. Although originally intended to be used on Web resources, RDF is capable of representing information about things that can be identified on the Web, even when they cannot be directly retrieved from the Web [15]. This capability makes RDF an appropriate metadata schema language for describing information related to the various results and outcomes of natural-disaster research. RDF is intended for situations in which information needs to be processed by applications, rather than only being displayed to people. RDF provides a common framework for expressing information and can thus be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information can be made available to applications other than those for which it was originally created [15].

Jena: Jena is a Java toolkit for building Semantic Web applications [1]. The *Jena Java RDF API* is developed by HP Labs for creating and manipulating RDF metadata. It mainly comprises:1)The "Another RDF Parser" (ARP), a streaming parser suitable for validating the syntax of large RDF documents. 2)A persistence subsystem, which provides persistence for RDF metadata through the use of a back-end database engine. It supports RDQL queries. 3)The RDF query language (RDQL), which is an implementation of an SQL-like query language for RDF. Jena generates RDQL queries dynamically and executes RDQL queries over RDF data stored in the relational persistence store.

Open Grid Service Architecture: Grid supports the sharing and coordinated use of diverse resources in dynamic, distributed "Virtual Organizations" (VOs)[**?**]. The Open Grid Services Architecture (OGSA) is a service-oriented Grid computing architecture, which an extensible set of Grid services that may be aggregated in various ways to meet the needs of VOs[**?**]. OGSA defines uniform Grid service semantics and standard mechanisms for creating, naming, and discovering Grid services. Web service technologies, such as XML, SOAP, WSDL, UDDI,etc., are adopted to build up the Grid services infrastructure. A Grid service is a Web service that provides a set of well-defined interface and that follows specific conventions [**?**]. The interface and behaviors of all Grid services is described by GWSDL[18].

3 Metadata Elicitation

Metadata is structured data about data. The goal of the gDisDL system is to support the storage and retrieval of metadata that pertain to a variety of results derived from



(a) Class Hierarchy for the EU-MEDIN RDF Schema

(b) Example of an EU-MEDIN RDF Schema

Fig. 1. EU-MEDIN RDF Schema

research in natural disasters (earthquakes, floods, forest fires, industrial hazards, landslides, volcano eruptions, etc). To this end, we need a metadata language defined in a common and open format, that will: (i) Promote the standardization of natural disaster metadata, while at the same time allowing future extensions; (ii) Enable the storage of metadata in different platforms according to a common, standard schema; (iii) Support the interoperability of different metadata repositories; in particular the specification of queries and the execution thereof upon different metadata databases.

We consider artifacts derived from natural-disaster research projects as resources whose properties will be represented in RDF. To specify the metadata for those artifacts, we conducted a detailed requirements analysis and came up with a classification containing 17 distinct resource classes: “EC project,” “Event,” “Journal paper,” “Software,” “Student Thesis (MSc or PhD),” “Other scientific paper,” “Report/deliverable,” “Web site,” “Press article,” “Book,” “Other project,” “Field experimental dataset,” “Laboratory experimental dataset,” “Spatial digital dataset,” “Hardware,” “Media presentation,” and “Unclassified activity.” The properties of and the relationships between class instances were defined accordingly.

We used the RDF Schema [9] to describe the identified set of RDF classes, properties, and values. The resulting schema is called EU-MEDIN RDF schema and represents our proposed metadata standard for natural-disaster research resources. Figure 1(a) shows the class hierarchy of the EU-MEDIN RDF schema.

Below, we give an example extracted from the EU-MEDIN RDF schema. This excerpt of our schema includes two classes, *Press article*, and *EC Project*, and three properties, *author, belongTo* and *name*. Using those classes and properties, we can describe in RDF the following piece of knowledge: “John Smith wrote a Paper, which belongs to FloodSim project. The paper’s title is “Flood Simulation on the Grid” in RDF. Figure 1(b) shows part of the schema definition and the RDF description of the example, presented as an RDF graph.

4 gDisDL System Design

gDisDL system is a Grid service-oriented system, which consists of the following components: the *Data Aggregator*, the *Searcher*, the *gDisDL Store*, and the *Data Manager*. As shown in Figure 2, the *gDisDL* system comprises a number of geographically distributed *gDisDL* nodes. Each node should includes a *Data Aggregator*, a *Searcher*, and a *gDisDL Store*. The *Data Aggregator* collects, validates, and encodes metadata in RDF; the *Searcher* is designed for querying RDF metadata; the *gDisDL Store* is used to store RDF metadata; the *Data manager* manages the RDF metadata maintained in the gDisDL stores. Finally, the *Editor* and the *Searcher GUI* are client tools enabling users to interacting easily with gDisDL through a graphical-user interface.

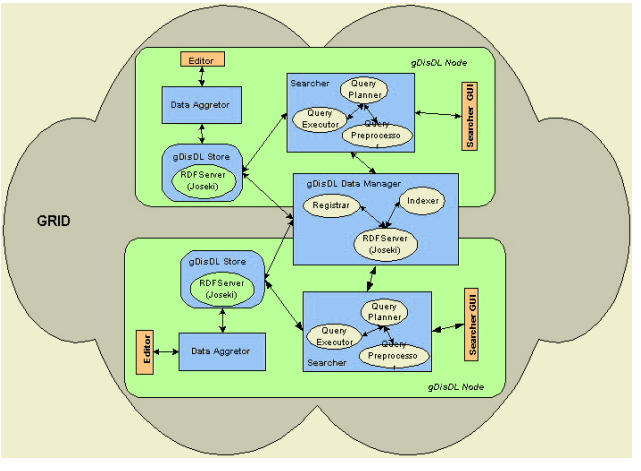


Fig. 2. The Architecture of the gDisDL System

4.1 Design Goals

The main design issue of the gDisDL system is how to share RDF metadata efficiently and securely in a distributed manner. To address this challenge, we adopt the Open Grid Service Architecture (OGSA). To this end, we design gDisDL as a set of Grid services, which are defined according to the Open Grid Service Infrastructure (OGSI) specifications.

Another design challenge is how to encode and store metadata in RDF. Currently, most RDF-based systems ask users to feed RDF metadata directly [4]. Therefore, users have to encode resource metadata into RDF syntax manually, a process which is inconvenient and difficult. To cope with this problem, we designed and implemented the *DataAggregator*, a component which generates the RDF syntax automatically and subsequently stores the RDF-encoded metadata in RDF storage.

Another challenge is the storage and query of RDF metadata. Typically, an RDF database can be used to store RDF metadata, and an RDF query language can be used

to express queries and execute them on the database. In the EU-MEDIN use-case scenario, however, most projects do not contribute large bodies of metadata; also, metadata updates are not very frequent. Therefore, a database system would be an overly expensive solution for our system requirements. Moreover, we would have to choose one among several existing RDF databases and query languages [4, 1, 17, 13], and integrate it with gDisDL. Thus, the system would depend heavily on the chosen database system. Currently, the default behavior of gDisDL is to store RDF metadata in plain files. In order to make gDisDL more open and extensible, we designed the *Searcher* component as a "query translator" that processes user requests and generates queries according to the back-end storage used in each gDisDL site. Thus, the back-end storage system remains transparent to the user and any RDF database systems can be deployed with gDisDL.

Another important design consideration is security. The security of the gDisDL is mainly concerning two aspects: one is accessing distributed RDF metadata secretly; another is that the shared RDF metadata should be protected, for instance, restricting the users who have not the right to access the information. We adopt the Grid security infrastructure (GSI) in the *gDisDL*, which uses the security mechanisms such as authentication and authorization to secrete the shared RDF metadata.

4.2 gDisDL Components

The Data Aggregator. The *Data Aggregator* encodes information provided by end-users for EU-MEDIN resources into RDF. In particular, the Aggregator:

1. Gets the information describing an EU-MEDIN resource from the *Editor*.
2. Validates the provided information with respect to the EU-MEDIN RDF schema, taking into account resource classes, class properties, restrictions, and data types.
3. If the information is deemed valid, the Aggregator will generate a unique URI to identify the resource. The URI contains two parts: one is the location of the gDisDL system that the user uses (e.g. the domain name); the other is the time when the RDF metadata was generated.
4. If the data is not valid, the Aggregator sends an error message to the Editor, and ends the process.
5. The Data Aggregator transforms the validated data together with the created URI into an RDF graph, which is a collection triples, each consisting of a subject, a predicate and an object [14]. The RDF metadata of the resource is thus created.
6. The RDF metadata is encoded in RDF/XML format and saved in a gDisDL store.

The gDisDL Store. *gDisDL Stores* are the repositories used to save RDF metadata created by the Data Aggregator. A gDisDL Store, by default, stores metadata for similar EU-MEDIN resources (i.e, resources belonging to the same EU-MEDIN class) into the same RDF file. For example, all RDF metadata describing Journal Papers are kept in one file, all RDF metadata describing Data Sets are kept in another file, and so on. Thus, when we look for metadata about Journal Papers, we can search into local or remote RDF files dedicates to Journal-paper metadata. In order for a gDisDL Store to make its contents available on the Grid, it has to register with one gDisDL Data Manager. Furthermore, the Store has to notify this Data Manage about updates in its contents. The Store's contents are published through Jena's Joseki Web server [1, 2].

The Searcher. The *Searcher* is the component responsible for the metadata search process. It comprises three subcomponents: the Query Pre-processor, the Query Planner, and the Query Executor. The Query Pre-processor receives and validates the user's requests using a mechanism similar to that of the Data Aggregator. The Query Planner works together with a Data Manager to prepare a query plan; the Query Executor executes the query according to the query plan.

A typical searching procedure begins with the Query Pre-processor that receives the request from the client. It checks the data of the request according to the RDF Schema, translates it into an RDF triple model, and passes the result to the Query Planner. The Query Planner then makes a query plan by consulting the gDisDL Data Manager. The query plan specifies what kind of back-end storage system should be queried, e.g. a gDisDL Store or an RDF database; in the first case, it also provides the address of the target *gDisDL Store*. Eventually, the Query Executer performs the real query according to the query plan. It can search the RDF metadata looking through the RDF file in target gDisDL Stores.

The query of the gDisDL system is based on the RDF triple model. Namely, the query is a a single triple pattern, with optional subject (parameter "s"), predicate (parameter "p"), and object (parameter "o"). Absence of a parameter implies "any" for matching that part of the triple pattern. The *Searcher* handles RDF data based on the triple (Subject, predicate, Object). According the user's input parameters, the *Searcher* will make a query and locate the RDF file that may contain the desired RDF metadata from the gDisDL store; then it explores all the RDF triple statements, e.g., the name or URI of the resources ("s"), their properties ("p"), and their values ("o"), compares them with the input parameters, and retrieves the matched RDF triples. if the back-end store is an RDF database system, the Query Planner can translate the user's input (i.e. the resource type, the property, values of the properties) into a proper RDF query language format.

The gDisDL Data Manager. The gDisDL Data Manager provides index information about RDF metadata maintained in distributed gDisDL Stores. To this end, it keeps a list of URI's of the stored RDF resources. The index information is used when making a query plan. The gDisDL Data Manager has three subcomponents: an Indexer, which maintains the list of URIs of all the metadata in all the gDisDL nodes; a Registrar for registering gDisDL Stores; the Joseki RDF web server, for retrieving and publishing RDF metadata, i.e., index information. The gDisDL Data Manager receives update notifications from its associated gDisDL Stores and updates its index accordingly. Each gDisDL store should register to the gDisDL Data Manager in order to share the RDF metadata of it.

5 Implementation

We have implemented a prototype of gDisDL. In this section, we provide some details about the gDisDL implementation.

The Grid gDisDL is implemented within the Open Grid Services Infrastructure (OGSI). Globus Toolkit 3 and Jena are the main development tools used in our im-

plementation. GT3 is a software toolkit that can be used to program grid-based applications. It is implemented in Java based on the OGSI specification. GT3 provides lot of services, programs, utilities, etc. Jena is a Java API that can be used to create and manipulate RDF graphs. it comprises object classes to represent graphs, resources, properties, and literals; a graph is called a model and is represented by the model interface. In our implementation, Jena is used as a JAVA RDF toolkit for creating, manipulating and querying RDF metadata.

To implement a Grid service, the first and most important work is to define the interface of the Grid service, namely, specify the service interface in GWSDL [18]. Once the interface is correctly defined in GWSDL, implementing the service using Java and other tools is straightforward. Thus we will focus on describing how the interface is defined and what kinds of operations will be invoked.

5.1 Data Aggregator Grid Services and Interface

The *DataAggregator service* processes the collected information and data from the *Editor* client, encodes it into RDF format, and saves it into gDisDL stores as RDF files. The interface of the DataAggregator grid service is defined in GWSDL as follows:

```
<gwsdl:portType name="DataAggregatorPortType" extends="ogsi:GridService">
  <operation name="retriveInfo">
    <input message="tns:GetInputMessage"/>
    <output message="tns:GetOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="getRDF">
    <input message="tns:GetRDFInputMessage"/>
    <output message="tns:GetRDFOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="validate">
    <input message="tns:ValInputMessage"/>
    <output message="tns:ValOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="saveRDF">
    <input message="tns:SaveInputMessage"/>
    <output message="tns:SaveOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
</gwsdl:portType>
```

Operation/PortType: `retrieveInfo()` is used to get values of the RDF triples from a client; `getRDF()` creates an RDF graph and assigns the values of the triples; `validate()` checks and validates the input data according to the syntax of EU-MEDIN RDF metadata; `saveRDF()` saves the RDF metadata into a file in RDF/XML syntax.

5.2 Searcher Grid Services and Interface

The Searcher grid service is used for receiving and answering user queries about EU-MEDIN resources. There are two cases for the Searcher when searching for RDF meta-

data. In the first case, the Searcher uses the RDF metadata document `match()` method to search the RDF metadata in a RDF document. In the second case, the search in conducted upon an RDF database, using a database-specific plug-in. Currently we just implemented the first one. The interface is defined as follows:

```
<gwsdl:portType name="SearcherPortType" extends="ogsi:GridService">
  <operation name="preprocess">
    <input message="tns:PreInputMessage"/>
    <output message="tns:PreOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="searchList">
    <input message="tns:ListInputMessage"/>
    <output message="tns:ListOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
  <operation name="match">
    <input message="tns:MatchInputMessage"/>
    <output message="tns:MatchOutputMessage"/>
    <fault name="Fault" message="ogsi:FaultMessage"/>
  </operation>
</gwsdl:portType>
```

Operation/PortType: the `preprocess()` operation is used for preprocessing user requests. The `searchList()` gets the remote RDF metadata information from a gDisDL Data Manager. The `match()` operation is used by Query Executor to match RDF triples.

5.3 gDisDL GUIs

Two graphical interfaces are designed and developed to facilitate the end user: the *Editor* and the GUI Client of the Searcher. The *Editor* is a GUI client of the Data Ag-

Fig. 3. GUI of gDisDL Searcher

gregator Grid service where a user can input information and data about EU-MEDIN resources. Similar to the EU-MEDIN portal, it also provides some forms which can be used to collect information about the EU-MEDIN resources. The Searcher GUI Client of the Searcher Grid Service is needed for users to input the parameters of metadata queries and get results (see Figure 3). The GUI allows users to specify the resource type, the property, values of the properties, etc. The GUI Client also decodes the RDF query results into human-readable form, and displays it on the result window (see Figure3).

6 Conclusions and Future Work

In this paper, we present an RDF-based Grid service approach for organizing and capitalizing European research results in the field of natural disasters. In brief, our approach makes the RDF metadata of the European research results in the field of natural disasters to be shared securely and effectively in a heterogeneous network environment using Grid technology. Then we describe the design and the prototype implementation of the Grid-enable gDisDL system. The RDF-based Grid-enable gDisDL system is a platform independent system which provides good interoperability with other systems. It can store, manage, and query RDF metadata in a secure and distributed manner.

Next step, we will develop a RDF database plug-in of the Searcher in order to integrate RDF databases into our system. A mechanism is also needed for the gDisDL Data managers to exchange indexing information. Furthermore, the gDisDL Data manager will be able to be used as a kind of Cache to retrieve and publish metadata located in the gDisDL stores for improving the query performance.

References

1. Jena - a semantic web framework for java. http://jena.sourceforge.net, 2003.
2. Joseki. http://www.joseki.org, 2003.
3. Eu-medin portal. http://www.eu-medin.org, 2004.
4. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proceedings of the Second International Workshop on the Semantic Web (SemWeb '01)*, pages 1–13, May 2001. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-40/.
5. Seaborne Andy. An rdf netapi. Technical report, HP Labs, 2002.
6. T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI):Generic Syntax*. The Internet Engineering Task Force, August 1998.
7. Tim Berners-Lee. *Notation 3*, 1998.
8. T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0*. World Wide Web Consortium, third edition, February 2004.
9. Dan Brickley and R.V. Guha. *Resource Description Framework (RDF) Schema Specification 1.0*. World Wide Web Consortium.
10. D. Connolly, F.V. Harmelen, I. Horrocks, D.L. McGuinness, P. Patel-Schneider, and L.A. Stein. *DAML+OIL (March 2001) Reference Description*. World Wide Web Consortium, March 2001.

11. I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.
12. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.
13. G.Karvounarakis, V.Christophides S.Alexaki, and Michel Scholl D.Plexousakis. RQL: A Declarative Query Language for RDF. The Eleventh International World Wide Web Conference (WWW'02), May 2004.
14. Graham Klyne and Jeremy Carroll. Resource Description Framework (RDF): Concepts and Abstract Data Model. Technical report, The World Wide Web Consortium, August 2002.
15. F. Manola and E. Miller (editors). RDF Primer. W3C Working Draft, October 2003. http://www.w3.org/TR/rdf-primer/.
16. P.F. Patel-Schneider, P. Hayes, and I. Horrock. *OWL Web Ontology Language Semantics and Abstract Syntax*. World Wide Web Consortium, February 2004.
17. Andy Seaborne. *RDQL - A Query Language for RDF*. The World Wide Web Consortium (W3C), January 2004.
18. Borja Sotomayor. The globus toolkit 3 programmer's tutorial. Technical report, The Globus Alliance, 2003.
19. S. Tuecke, I. Foster K. Czajkowski, C. Kesselman J. Frey, S. Graham, T. Sandholm T. Maguire, and D. Snelling P. Vanderbilt. Open grid service infrastructure (ogsi) version 1.0. *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.

Optimising Parallel Applications on the Grid Using Irregular Array Distributions*

Radu Prodan and Thomas Fahringer

Institute for Computer Science, University of Innsbruck,
Technikerstraße 13, A-6020 Innsbruck, Austria
{radu, tf}@dps.uibk.ac.at

Abstract. In this paper we propose a new approach for scheduling data parallel applications on the Grid using irregular array distributions. We implement the scheduler as a new case study for using a general purpose experiment management tool that we developed in previous work for performance tuning and optimisation of scientific applications. We report results on scheduling a Jacobi relaxation application on a simulation testbed of the Austrian Grid [2] using a problem independent plug-and-play genetic algorithm.

1 Introduction

Even though the message passing paradigm used for writing tightly-coupled parallel applications contradicts with the loosely-coupled Grid model, it is currently still being used in the community as an opportunity for executing large problem sizes of existing parallel applications on the Grid. In this context, scheduling parallel applications relies on appropriate mappings of the parallel processes onto the heterogeneous Grid resources. This often requires *irregular data distributions* such that more work is assigned to the processes mapped onto the fast machines.

Related work on application-level scheduling [3] based on the explicit message passing-based paradigm (often called fragmented programming) designs a generic data mapping module, but omits to concretely specify how this difficult problem is realised for message passing programs.

In this paper we propose the High Performance Fortran (HPF) [8] data distribution model as an explicit method to specify irregular data distributions for scheduling data parallel applications on the Grid. We instantiate our approach as a new case study of using a Grid-enabled experiment management tool that we designed in previous work for cross-experiment performance analysis and optimisation of scientific applications [10].

In the next section we give a short background on our experiment management and optimisation tool. Section 3 introduces the irregular array distribution model and the parameterisation technique that we use to solve the scheduling

* This research is supported by the Austrian Science Fund as part of the Aurora project under contract SFBF1104.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 527–537, 2005.

problem. Section 4 presents a simple prediction model for a Jacobi relaxation method that we use as the sample objective function in our experiments. Section 5 presents experimental results on tuning a genetic scheduling algorithm to the Austrian Grid [2] resource characteristics. Section 6 concludes the paper.

2 ZENTURIO Optimisation Framework Design

ZENTURIO [10] is an experiment management tool for automatic cross-experiment performance and parameter studies of parallel applications on cluster and Grid architectures. A *directive-based language* (in the Fortran HPF and OpenMP style) called ZEN is used to specify value ranges for arbitrary application parameters, including problem and machine sizes, loop and array distributions, software libraries and algorithms, interconnection networks, or execution machines (see Examples 1, 2, and 3). *Constraint directives* are used to filter the meaningless experiments from the default cross product of the parameter values (see the directives $c1$ and $c2$ in Example 1). Additionally, *performance directives* are used for measuring a wide range of performance metrics (e.g., execution, communication, synchronisation times, hardware counters, parallelisation overhead) focused to arbitrary code regions (see the directive p in Example 1).

After the user parameterised the application with ZEN directives (which define a problem independent search space), ZENTURIO automatically instruments, generates, and executes the experiments. The suite of experiments is generated either *exhaustively*, or according to a general-purpose *heuristic algorithm*, like subdivision, simplex, simulated annealing, BFGS, EPSOC, or genetic algorithms. The heuristic algorithm tries to find an experiment that maximises an input objective function that implements a problem independent interface (e.g., ZEN performance metric for performance tuning or analytical prediction function in case of scheduling). After the completion of each experiment, the performance and the output files are automatically stored into a relational data repository. An visualisation portal allows the user to automatically query the database by mapping the application parameters and the performance metrics to the axis of a variety of visualisation diagrams, including linechart, barchart, piechart, and surface (see Figure 4 in Section 5).

3 Irregular Array Distributions

High Performance Fortran (HPF) [8] has been designed to express array distributions at a high-level of abstraction, while offering the programmer a single program view which is not fragmented by low-level message passing library routines. In this paper we propose a new formal mechanism for scheduling data parallel applications on heterogeneous Grid resources using irregular array distributions. We use the Vienna Fortran Compiler [1] to translate an HPF application into an MPI equivalent that we link with the Grid-enabled MPICH-G2 [6] library for Grid execution.

3.1 General Block Distribution

Let $\texttt{MAT}(m, n)$ denote a two-dimensional matrix and $\texttt{PROC}(p, q)$ a two-dimensional processor array.

Definition 1. *Let $Bx(p)$ and $By(q)$ denote two one-dimensional arrays that satisfy the conditions: $\sum_{i=1}^{p} Bx_i \geq m$ and $\sum_{i=1}^{q} By_i \geq n$. The* general block data distribution *of* ***MAT*** *using the mapping arrays Bx and By is a function:*

$$\textit{DISTR} : [1..m] \times [1..n] \rightarrow [1..p] \times [1..q],\ \textit{DISTR}(x, y) = (z, w),$$

such that: $\sum_{i=1}^{z-1} Bx_i < x \leq \sum_{i=1}^{z} Bx_i,\ \forall\ x \in [1..p]$ and $\sum_{i=1}^{w-1} By_i < y \leq \sum_{i=1}^{w} By_i,\ \forall\ y \in [1..q]$. The partition:

$$\textit{MAT}_{\textit{PROC}(i,j)} = \{\textit{MAT}_{k,l} \mid \textit{DISTR}(k, l) = (i, j),\ \forall\ k \in [0..p],\ \forall\ l \in [0..q]\}$$

is called the distribution *of* ***MAT*** *onto the processor $\textit{PROC}_{ij}$.*

Example 1 (Parameterised HPF general block array distribution).

```
p:  !ZEN$ CR CR_A PMETRIC WTIME, ODATA
          INTEGER, PARAMETER m = 4, n = 8, p = 2, q = 3
          REAL MAT(m, n)
    !HPF$ PROCESSOR PROC(p, q)
          INTEGER, PARAMETER :: x1 = 3, x2 = 1, y1 = 2, y2 = 2, y3 = 4
    !ZEN$ SUBSTITUTE x1 = { 0 : 4 } BEGIN
    !ZEN$ SUBSTITUTE x2 = { 0 : 4 } BEGIN
    !ZEN$ SUBSTITUTE y1 = { 0 : 8 } BEGIN
    !ZEN$ SUBSTITUTE y2 = { 0 : 8 } BEGIN
    !ZEN$ SUBSTITUTE y3 = { 0 : 8 } BEGIN
          INTEGER, PARAMETER :: Bx(p) = (/ x1, x2 /)
          INTEGER, PARAMETER :: By(q) = (/ y1, y2, y3 /)
    !ZEN$ END SUBSTITUTE
          . . .
    !HPF$ DISTRIBUTE MAT(GEN_BLOCK(Bx), GEN_BLOCK(By)) ONTO PROC
c1: !ZEN$ CONSTRAINT VALUE x1 + x2 == 4
c2: !ZEN$ CONSTRAINT VALUE y1 + y2 + y3 == 8
```

Example 1 defines one matrix $\texttt{MAT}(m, n)$ which has both dimensions distributed over the processor array $\texttt{PROC}(p, q)$ using the general block mapping arrays $Bx(p)$ and $By(q)$ (see Figure 1). One can notice that the HPF and MPI programming paradigms (improperly) consider the Grid as a single parallel computer. The HPF `PROCESSORS` directive in this approach refers to the complete set of Grid machines retrieved from the Grid information service [5] and organised in a two-dimensional array $\texttt{PROC}(p, q)$ constrained to the cardinality of $p \cdot q$.

A general block array distribution defines an application *schedule*. We specify the complete set of possible distributions by representing each mapping array element (i.e., $Bx_i, \forall\ i \in [1..p]$ and $By_j, \forall\ j \in [1..q]$) as a generic problem parameter. For this reason, we encode the default elements of the mapping arrays $Bx(p)$

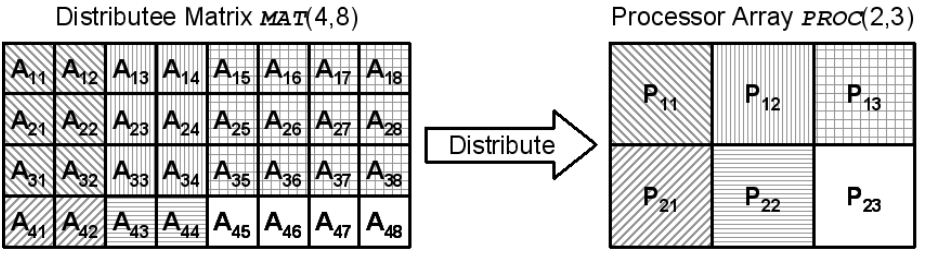


Fig. 1. The default general block array distribution defined in Example 1

and $By(q)$ as program constants. We annotate each such parameter with a ZEN directive that defines the complete set of possible block size instantiation values (i.e. from 0 to m or n, depending on the distribution axis). A distribution of size zero on one processor (i.e., Grid machine) influences the application machine size, since the corresponding processor will not take part in the computation. The constraint directives insure that the sum of the general block mapping elements match the matrix rank on each dimension. The ZEN directive parameter annotations define a search space of size $(m+1)^{p-1} \cdot (n+1)^{q-1}$ of possible matrix schedules (two orders of magnitude are eliminated by the two constraints).

3.2 Indirect Distribution

The same techniques presented in the previous section can be applied for the more general indirect array distribution. Let $\texttt{MAT}(m, n)$ denote a two-dimensional matrix and $\texttt{PROC}(p, q)$ a two-dimensional processor array.

Definition 2. *Let $I(p, q)$ denote a two-dimensional distribution array such that $I(i, j) \leq p \cdot q,\ \forall\ i \in [1..m],\ \forall\ j \in [1..n]$. The* indirect data distribution *of MAT using the mapping matrix I is a function:*

$$\mathit{DISTR} : [1..m] \times [1..n] \rightarrow [1..p] \times [1..q],\ \mathit{DISTR}(x, y) = \left(I(x, y)\ mod\ p,\ \left[\frac{I(x, y)}{p} \right] \right).$$

Example 2 defines the matrix $\texttt{MAT}(m, n)$ whose elements are indirectly distributed across the over processor array $\texttt{PROC}(p, q)$ (representing again the entire Grid) according to the mapping matrix $I(m, n)$ (see Figure 2). The default elements of the mapping matrix $I(m, n)$ are program constants that represent the processors where each array element shall be distributed. We represent each mapping array element as an application parameter

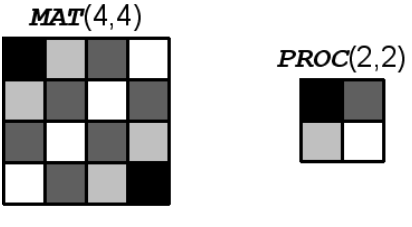


Fig. 2. The default indirect array distribution defined in Example 2

annotated with a ZEN directive that specifies the complete set of possible indirect distributions. These ZEN directives define a search space of size $(p \cdot q)^{mn}$ of possible indirect array distributions (i.e., schedules). This search space in this case, however, tends to be rather huge compared to general block, as it exponentially depends on the matrix size.

Example 2 (Parameterised HPF indirect array distribution).

```
      INTEGER, PARAMETER m = 4, n = 4, p = 2, q = 2
      DIMENSION MAT(m, n)
!HPF$ PROCESSORS PROC(p, q)
      INTEGER, PARAMETER M11 = 1, M12 = 2, M13 = 3, M14 = 4
      INTEGER, PARAMETER M21 = 2, M22 = 3, M23 = 4, M24 = 2
      INTEGER, PARAMETER M31 = 3, M32 = 4, M33 = 3, M34 = 2
      INTEGER, PARAMETER M41 = 4, M42 = 3, M43 = 2, M44 = 1
!ZEN$ SUBSTITUTE M11 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M12 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M13 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M14 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M21 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M22 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M23 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M24 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M31 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M32 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M33 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M34 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M41 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M42 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M43 = { 1 : 4 } BEGIN
!ZEN$ SUBSTITUTE M44 = { 1 : 4 } BEGIN
      INTEGER I(m,n) = (/ (/ M11, M12, M13, M14 /),
                          (/ M21, M22, M23, M24 /),
                          (/ M31, M32, M33, M34 /),
                          (/ M41, M42, M43, M44 /) /)
!ZEN$ END SUBSTITUTE
      . . .
!HPF$ DISTRIBUTE MAT(INDIRECT(MAP)) ONTO PROC
```

4 Objective Function

Application scheduling relies on performance prediction models, which is a difficult research topic on its own that goes beyond the optimisation and scheduling topics addressed in this paper. Therefore, we exemplify our techniques in the context of a Jacobi relaxation application for which we could formulate simple analytical prediction formulas.

Let $\mathcal{E}$ denote an experiment of a parameterised application that implements a two-dimensional Jacobi relaxation method as sketched in the Examples 1 and 2

and $\mathcal{E}\left(\texttt{MAT}_{\texttt{PROC}(i,j)}\right)$ the parallel process that that hosts the (irregular) distribution of MAT onto the processor $\texttt{PROC}_{i,j}$ (see Definition 1). Let $\texttt{COMP}(\mathcal{E})$ denote the computation performance metric, $\texttt{COMM}(\mathcal{E})$ the data communication metric of $\mathcal{E}$, and $\mathcal{M}(\texttt{PROC}_{i,j})$ the physical memory available on the processor $\texttt{PROC}_{i,j}$ as recorded by the Grid information service [5]. We approximate the *objective function* for the parallel Jacobi relaxation as the maximum sum between the computation and the communication metrics across all the parallel processes:

$$\mathcal{F}(\mathcal{E}) = \begin{cases} \underset{\substack{\forall i \in [1..p] \\ \forall j \in [1..q]}}{max} \left\{ \texttt{COMP}\left(\mathcal{E}\left(\texttt{MAT}_{\texttt{PROC}(i,j)}\right)\right) + \texttt{COMM}\left(\mathcal{E}\left(\texttt{MAT}_{\texttt{PROC}(i,j)}\right)\right) \right\}, \\ \qquad \textit{if } \left|\texttt{MAT}_{\texttt{PROC}(i,j)}\right| \cdot \textit{size(e)} \leq \mathcal{M}(\texttt{PROC}_{i,j}),\ \forall i \in [1..p] \wedge \forall j \in [1..q]; \\ \infty, \qquad \textit{if } \exists i \in [1..p] \vee \exists j \in [1..q], \left|\texttt{MAT}_{\texttt{PROC}(i,j)}\right| \cdot \textit{size(e)} > \mathcal{M}(\texttt{PROC}_{i,j}), \end{cases}$$

where $\left|\texttt{MAT}_{\texttt{PROC}(i,j)}\right|$ is the cardinality of the distribution of MAT onto the processor $\texttt{PROC}_{i,j}$ and *size(e)* is the size in bytes of a matrix element e.

In the case of the general block array distribution, the computation and the communication metrics could be analytically approximated as:

$$\begin{aligned} &\texttt{COMP}(\mathcal{E}(\texttt{MAT}_{\texttt{PROC}(i,j)})) = Bx_i \cdot By_j \cdot \frac{W_e}{v_{\texttt{PROC}(i,j)}} \cdot I; \\ &\texttt{COMM}(\mathcal{E}(\texttt{MAT}_{\texttt{PROC}(i,j)})) = \overline{\mathcal{L}(\texttt{PROC}_{i,j})} + \\ &\qquad + Bx_i \cdot \textit{size(e)} \cdot \left(\frac{1}{\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{i-1,j})} + \frac{1}{\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{i+1,j})} \right) + \\ &\qquad + By_j \cdot \textit{size(e)} \cdot \left(\frac{1}{\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{i,j-1})} + \frac{1}{\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{i,j+1})} \right), \end{aligned}$$

$\forall\ i \in [1..p],\ \forall\ j \in [1..q]$, where:

- W_e is the work required to compute one matrix element e, expressed in floating point instructions (i.e., four in case of the Jacobi application);
- $v_{\texttt{PROC}(i,j)}$ is the speed of the Grid machine $\texttt{PROC}_{i,j}$ in floating point instructions per second (e.g., as measured using the LINPACK benchmark);
- I is the number of iterations;
- $\overline{\mathcal{L}(\texttt{PROC}_{i,j})}$ is the total latency required for receiving the four neighbouring matrix elements: $\overline{\mathcal{L}(\texttt{PROC}_{i,j})} = \mathcal{L}(\texttt{PROC}_{i,j}, \texttt{PROC}_{i-1,j}) + \mathcal{L}(\texttt{PROC}_{i,j}, \texttt{PROC}_{i+1,j}) + \mathcal{L}(\texttt{PROC}_{i,j}, \texttt{PROC}_{i,j-1}) + \mathcal{L}(\texttt{PROC}_{i,j}, \texttt{PROC}_{i,j+1})$;
- $\mathcal{L}(\texttt{PROC}_{i,j}, \texttt{PROC}_{k,l})$ is the latency between the processors $\texttt{PROC}_{i,j}$ and $\texttt{PROC}_{k,l}$;
- $\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{k,l})$ is the bandwidth between the processors $\texttt{PROC}_{i,j}$ and $\texttt{PROC}_{k,l}$, as obtained from the Grid resource information service.

For the Jacobi relaxation application with an indirectly distributed matrix, the computation and the communication metrics could be approximated as:

$$\texttt{COMP}(\mathcal{E}(\texttt{MAT}_{\texttt{PROC}(i,j)})) = \left|\texttt{MAT}_{\texttt{PROC}(i,j)}\right| \cdot \frac{W_e}{v_{\texttt{PROC}(i,j)}};$$

$$\texttt{COMM}(\mathcal{E}(\texttt{MAT}_{\texttt{PROC}(i,j)})) = \sum_{\forall (k,l) \in \texttt{MAT}_{\texttt{PROC}(i,j)}} size(e) \cdot \texttt{COMM}(\texttt{MAT}_{k,l});$$

$$\texttt{COMM}(\texttt{MAT}_{k,l}) = \frac{1}{\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{\texttt{DISTR}(i-1,j)})} + \frac{1}{\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{\texttt{DISTR}(i+1,j)})} + \\ + \frac{1}{\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{\texttt{DISTR}(i,j-1)})} + \frac{1}{\mathcal{B}(\texttt{PROC}_{i,j}, \texttt{PROC}_{\texttt{DISTR}(i,j+1)})},$$

which ignores latencies for brevity reasons.

5 Parameter Tuning

In this section we report experiments on tuning a genetic algorithm for scheduling the Jacobi application within a simulation testbed of the Austrian Grid environment consisting of about 200 machines running Globus 2.4.2. We organised the fastest Grid machines into a square processor matrix of size 14×14. We have chosen a square matrix of size $10^4 \times 10^4$ distributed over the Grid using the general block distribution, and a fixed number of 100000 iterations. These inputs produce a search space of size: $(10^4+1)^{14-1} \cdot (10^4+1)^{14-1} \approx 10^{104}$ points.

We assume the user is familiar with genetic algorithm fundamentals [7]. To search for an optimal schedule using ZENTURIO, we plug-in a problem independent genetic search engine that encodes a generic application parameter defined by a ZEN directive (and represented in this instantiation by a general block mapping array element – see Example 1) as a *gene*. The complete string of genes builds a *chromosome*. A *schedule* is obtained by instantiating each gene with a concrete block size. We employ a classic *generational* genetic search algorithm that uses the reminder stochastic sampling with replacement *selection* mechanism and the single point *crossover* and *mutation* operators, as described in [7].

The quality of the solutions delivered by the genetic algorithm is heavily influenced by seven input parameters: population size, crossover and mutation probabilities, maximum generation number, steady state generation number, fitness scaling factor, and use of the elitist model. We automatically tuned the algorithm parameters as a scalability exercise for an exhaustive parameter tuning experiment using ZENTURIO. We specified the interesting parameter values for the genetic scheduler by annotating the PBS script used to submit the schedule experiments on a dedicated Beowulf cluster as shown in Example 3. The cross product of these parameters defines a total of 3840 experiments which were automatically conducted by ZENTURIO. Every experiment represents an instance of the genetic scheduling algorithm configured using a different parameter combination. All the experiments use the Grid resource information collected at the same time instance (i.e., Grid snapshot). This hierarchical experimental setup that applies ZENTURIO as an exhaustive cross-experiment performance study tool on the genetic search optimisation engine is depicted in Figure 3.

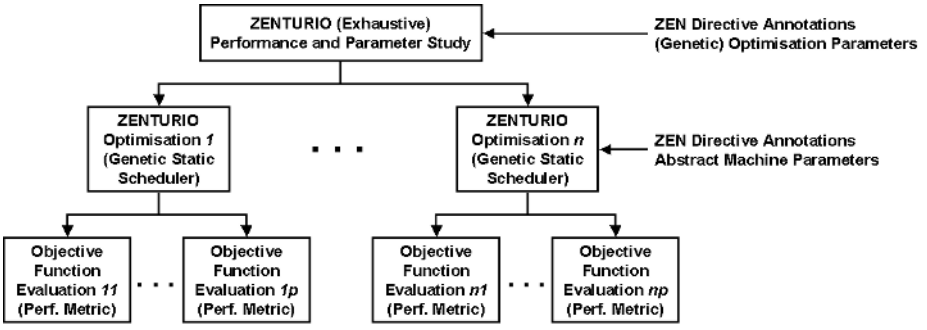
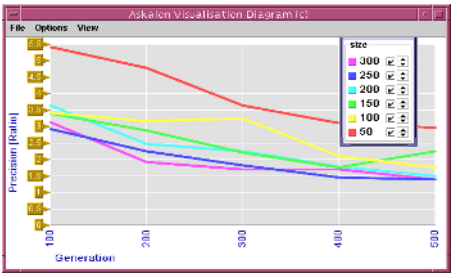


Fig. 3. Hierarchical experimental setup for genetic scheduling algorithm tuning

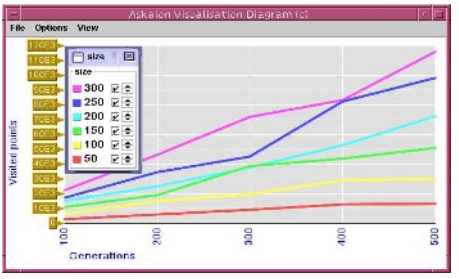
We evaluate the quality of the solutions produced by the algorithm against a set of 16 parallel $3GHz$ processors (that are twice as fast as any other machine in the Austrian Grid) that we know to deliver a good scalability for this problem size from a previous study. We refer to the execution time of the Jacobi application on this machine set as *optimal fitness* $\mathcal{F}_o$. From each experiment we collect three metrics that characterise the performance of the genetic algorithm: (1) *precision* P of the best individual $\mathcal{F}_b$ compared to the optimum $\mathcal{F}_o$, defined as the ratio: $P = \frac{\mathcal{F}_b}{\mathcal{F}_o}$; (2) *visited points* representing the total set of individuals (i.e., schedules) which have been evaluated by the algorithm during the search process; (3) *improvement* I in the fitness $\mathcal{F}_b$ of the last generation best schedule, compared to the first generation best schedule $\mathcal{F}_f$: $I = \frac{\mathcal{F}_f - \mathcal{F}_b}{\mathcal{F}_b} \cdot 100$. To attenuate the stochastic errors to which randomised algorithms are bound, we repeat each scheduling experiment for 40 times and report the arithmetic mean.

Example 3 (Parameterised PBS script).

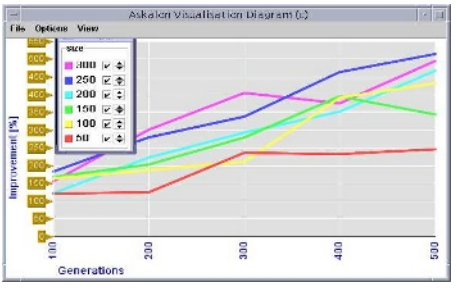
```
#PBS -l walltime=00:10:00:nodes=1:ppn=1
size = 300
#ZEN$ ASSIGN size = { 50 : 300 : 50 }
crossover = 0.9
#ZEN$ ASSIGN crossover = { 0.25 : 1 : 0.25 }
mutation = 0.001
#ZEN$ ASSIGN mutation = { 0.01, 0.05, 0.1, 0.2 }
generations = 500
#ZEN$ ASSIGN generations = { 100 : 500 : 100 }
convergence = 0.2
#ZEN$ ASSIGN convergence = { 0.1, 0.2 }
scaling = 2
#ZEN$ ASSIGN scaling = { 1, 2 }
elitist = T
#ZEN$ ASSIGN elitist = { T, F }
${JAVA} -DSIZE=${size} -DCROSSOVER=${crossover} ...
```



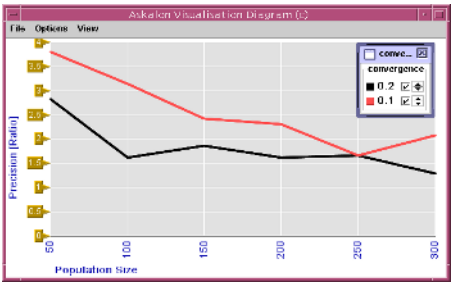
(a) Population Size.



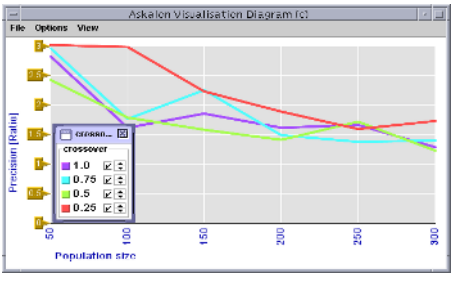
(b) Visited Points.



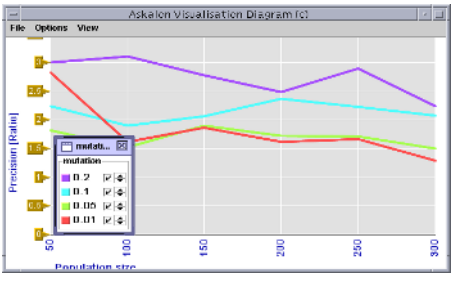
(c) Best Individual Improvement.



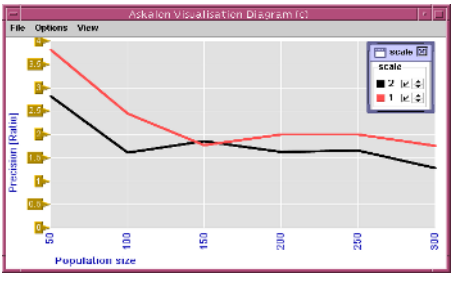
(d) Generation Percentage.



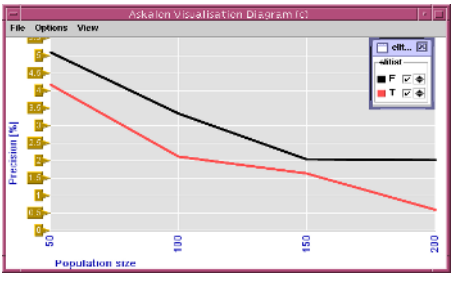
(e) Crossover Probability.



(f) Mutation Probability.



(g) Fitness Scaling Factor.



(h) Elitist Model.

Fig. 4. Genetic scheduling algorithm parameter tuning results

Various automatically generated diagrams from this experiment are depicted in Figure 4, which we cannot analyse in detail due to paper length constraints. The execution time of the genetic algorithm is proportional with the number of visited points (see Figure 4(b)). Considering $2ms$ per schedule evaluation on a $3GHz$ Intel Pentium processor, this translates to an average of $3min$ per genetic algorithm execution. In addition, genetic algorithms are well known to be embarrassingly parallel [9] which further decreases their execution time.

The genetic algorithm parameters depend most heavily on the search space characteristics, which is exclusively represented in this instantiation by the Austrian Grid testbed. Therefore, we are currently using the following genetic algorithm parameter configuration: population size: 300, crossover probability: 0.9, mutation probability: 0.001, maximum generation: 500, steady state generation percentage: 20%, fitness scaling factor: 2, elitist model: yes. In this configuration, our algorithm constantly produces about 25% precision and 500% improvement in solution, by visiting a fraction (i.e., 10^5 from 10^{104}) of the entire space points.

6 Conclusions

In this paper we showed a new application of an experiment management and optimisation tool for scheduling data parallel applications on the Grid using well-defined HPF irregular array distributions. In contrast to [3] which focuses on the explicit message passing paradigm, our scheduler has the advantage of clearly specifying and exposing to the user the application parameter space, represented in this instantiation by the complete set of possible array distributions, based on a single unified (i.e., not fragmented with message passing routines) program view. Our approach is, therefore, apart from the specification of the objective prediction function, application independent and, furthermore, not restricted to the particular scheduling problem addressed by this paper. A generic directive-based language used for parametrisation enables the realisation of problem independent heuristics, such as genetic algorithms, that could be plugged-in for other optimisation problems too. We have reported successful experimental results on scheduling a parallel Jacobi relaxation application using a genetic algorithm within a simulation testbed of the Austrian Grid. We are currently working on a new application of our tool for scheduling workflow applications in the frame of the ASKALON programming environment and tool-set for cluster and Grid computing [4].

References

1. S. Benkner. VFC: The Vienna Fortran Compiler. *Scientific Programming, IOS Press, The Netherlands*, 7(1):67–81, 1999.
2. The Austrian Grid Consortium. http://www.austriangrid.at.
3. Holly Dail, Henri Casanova, and Fran Berman. A Decoupled Scheduling Approach for the GrADS Program Development Environment. In *SC'2002 Conference CD*, Baltimore, November 2002. IEEE/ACM SIGARCH.

4. T. Fahringer. ASKALON - A Programming Environment and Tool Set for Cluster and Grid Computing. http://www.par.univie.ac.at/project/askalon, Institute for Computer Science, University of Innsbruck.
5. Steve Fitzgerald, Ian Foster, Carl Kesselman, Gregor von Laszewski, Warren Smith, and Steve Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, Portland, August 1997.
6. I. Foster and N. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *Proceedings of SC'98*. ACM Press, 1998.
7. David E. Goldberg. *Genetic Algorithms in Search, Optimization 6 Machine Learning*. Reading. Addison-Wesley, Massachusetts, 1989.
8. High Performance Fortran Forum. High Performance Fortran language specification. *Scientific Programming*, 2(1-2):1–170, 1993.
9. Yu-Kwong Kwok and Ishfaq Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 47(1):58–77, 25 November 1997.
10. Radu Prodan and Thomas Fahringer. ZENTURIO: A Grid Middleware-based Tool for Experiment Management of Parallel and Distributed Applications. *Journal of Parallel and Distributed Computing*, 64/6:693–707, 2004.

Dynamic Adaptation for Grid Computing

Jérémy Buisson[1], Françoise André[2], and Jean-Louis Pazat[1]

[1] IRISA/INSA de Rennes, Rennes, France
[2] IRISA/Université de Rennes 1, Rennes, France

Abstract. As Grid architectures provide resources that fluctuate, applications that should be run on such environments must be able to take into account the changes that may occur. This document describes how applications can be built from components that may dynamically adapt themselves. We propose a generic framework to help the developpers of such components. In the case of a component that encapsulates a parallel code, a consistency model for the dynamic adaptation is defined. An implementation of a restricted consistency model allowed us to experiment our ideas.

1 Introduction

Grid architectures differ from classical execution environments in that they are mainly built up as a federation of pooled resources. Those resources include processing elements, storage, network, and so on; they come from the interconnection of parallel machines, clusters, or any workstation. One of the main properties of these resources is to have changing characteristics even during the execution of an application. Resources may come and go; their capacities may vary during the execution of the applications. Moreover, resources may be allocated then reclaimed and reallocated as applications start and terminate on the Grid. Thus, resource usage by applications cannot be static; neither can changes in resource allocation be considered as faults. Grid-enabled application designers must keep in mind that resources and resource management are highly dynamic within Grid architectures.

Dynamic adaptation is a way to support evolving execution environments. It aims at allowing applications to modify themselves depending on the available resources. The key idea is that when the environment changes, the application should also change to fit with it. To react softly, the application can negotiates with its execution environment about resources, instead of undergoing the decisions of the execution environment. Because taking adaptation into account should not burden too much the application developpers, it is necessary to provide them adequate frameworks and mechanisms. That is our main research objective.

Section 2 makes a tour of existing researches in areas close to adaptation. Section 3 exposes how we model the dynamic adaptation of an application. Section 4 shows the architecture we are currently building as a support to make

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 538–547, 2005.

adaptable applications. Section 5 describes the consistency model we introduced for the adaptation of components that encapsulate parallel codes. Section 6 presents the state of the experiments we have done with our consistency model.

2 Adaptation Through Existing Works

Several projects use the word adaptation to describe themselves. As this section presents, all of those projects have different views of what adaptation is, what it consists in and why it should be used. Basically, the only common point is that adaptation consists in changing some things in applications. This asks four major questions: why adaptation should be done (section 2.1); where is it done in the execution (section 2.2); how can it be done (section 2.3); when should it be done (section 2.4). As the section shows, the several projects that exist have different answers to these four questions.

2.1 Goal of the Adaptation

With existing projects, two main views of what the goal of adaptation could be opposed. In [1], the Grid.It project and its ASSIST [2] programming model present adaptation as a way to achieve a specified level of performance. An application should modify its structure or change the resources it has allocated when performance contracts are not satisfied.

On the other hand, projects such as SaNS [3], GrADS [4] and PCL [5] consider that applications should adapt themselves to optimize themselves. Adaptation is a way for application to provide a "best effort" strategy. In this case, the application chooses the best implementation given the allocated resources and the properties of input data.

2.2 Location of the Adaptation

Many projects such as SaNS, GrADS and GrADSolve [6] do the adaptation upon invokation: when a function, method, procedure is called, the best algorithm is chosen. Although this is not as static as the automatic optimization such as for ATLAS [7], adaptation is not possible once the adaptable software has started its execution. This approach is realistic only if the adaptable softwares are fine-grained enough. In the case of SaNS, the targeted softwares are libraries of numerical kernels. In this case, it is sufficient to adapt only at invokation time.

Some projects such as Grit.It and PCL allows dynamic adaptation (namely, adaptation of a software that is executing). This is necessary when the execution time of the adaptable software is high. Moreover, the PCL project defines in [8] a consistency model for the adaptation of parallel codes.

2.3 Means of the Adaptation

When the adaptation is done exclusively at invokation time, adaptation is simply the selection of one implementation of several that are available. This is what is done with SaNS and GrADSolve.

In the case of dynamic adaptation, this is not sufficient. Adaptation either involves parameterizable softwares or reflexive programming. For example, within the ASSIST model of the Grid.It project, the *parmod* skeleton exposes several parameters such as the degree of parallelism that can be changed dynamically; the PCL project defines a framework for reflexive programming in distributed and parallel applications. Dynamic aspect weavers (such as Arachne [9]) may also be used as an alternative to reflexive programming to dynamically adapt the software.

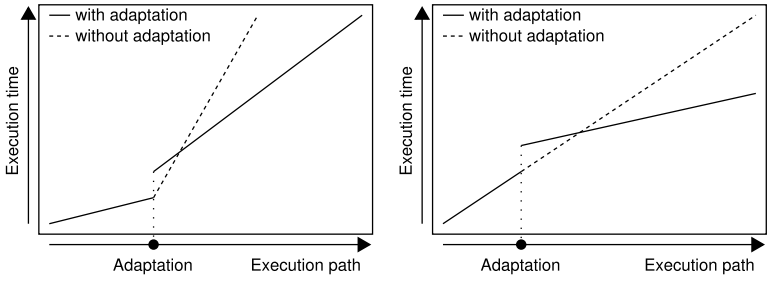
2.4 Decisions to Adapt

Several approaches can be used to decide when a software should adapt itself and what it should do. Within the ASSIST model, since adaptation is a way to achieve the contracted performance level, adaptation is mostly triggered from feedback control. On the other side, within the PCL project, the trigger is the reception of external events generated by external monitors. The projects SaNS and GrADSolve use performance models and a database of empirical measures to choose the right implementation to use.

3 Model of Dynamic Adaptation

Our model for dynamic adaptation considers that dynamic adaptation should occur in order to maximize the adequation between the application and its execution environment. Namely, this means that the purpose of dynamic adaptation is to optimize the application whenever its execution environment changes. To do so, it may use whatever means the developper is willing to.

The figure 1 shows the two kinds of curves for the progress of an adaptive software. The two cases correspond to an adaptation that causes the execution to slow down 1(a) and one that makes the application accelerates 1(b). The discontinuity reflects the cost of the adaptation. Although the adaptation optimizes



(a) Execution that is slowing down

(b) Execution that is speeding up

Fig. 1. Progress of an adaptive software

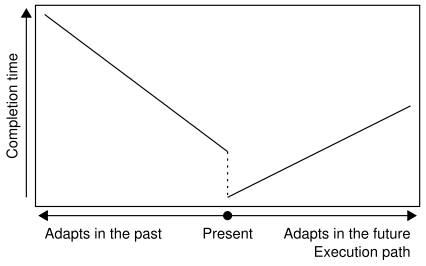


Fig. 2. Completion time depending on the state at which the adaptation is done

the application, the application may slow down. Indeed, this occurs when the execution environment reclaims some resources. Even if the application slows down, it is optimized compared to its execution if it has not been adapted: in such a case, the application could even have crashed.

The application can adapt itself anywhere in the execution path. It can be either at a past state or at a state in the future. The figure 2 shows how the overall completion time conceptually behaves depending on the current state and the one at which the adaptation is done. This curve directly translates that the adaptation optimizes the application. Thus, in the future, the sooner it is done, the better the completion time is. It reduces the time during which the application is suboptimal. By symmetry, in the past, the earlier in the application lifetime the adaptation is done, the bigger the computations to be redone are. Adapting "just after now" is better than "just before now" because of the overhead of restoring a past state.

Adapting in the past has the advantage over adapting in the future that the adaptation can occur immediatly once the decision has been made. Thus, there is no transition during which the execution is suboptimal. Moreover, it does not require the prediction of a point in the future of the execution path, which is an undecidable problem in the general case.

On the other hand, adapting in the future does not require any instruction to be reexecuted; neither it has the overhead due to checkpoint-taking.

4 Architecture of Adaptable Software

Applications are considered to be built as assemblies of software components. Each of those components can encapsulate sequential or parallel codes. Moreover, each component can adapt itself. To do so, the component is supported by a framework. Indeed, it appears that some of the mechanisms involved in adaptation are independant of the component itself.

4.1 Architecture of an Adaptable Component

The figure 3 shows the architecture of a parallel component. Five major functional boxes have been identified that lie in three parts of an adaptable component built with our adaptation framework.

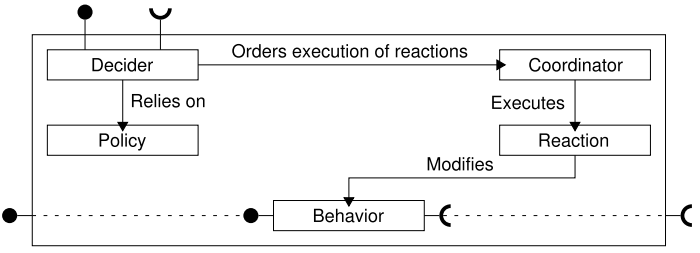


Fig. 3. Overall architecture of an adaptable component

- Functional part of the component.
  - **Behavior.** This is the implementation of the services provided by the component. An adaptable component is allowed to contain several behaviors that are alternative implementations.
- Component-specific part of the adaptation framework.
  - **Reaction.** This is a code that modifies the behavior that is being executed by the component. A component can include several reactions. The reactions can change some parameters of the behavior; replace the behavior with another one; modify the behavior with the help of reflexive programming or dynamic aspect weaving.
  - **Policy.** This provides all the necessary information that are specific to the component to make decisions concerning the adaptation. It eddicts on which events the component should adapt and which reaction it should use.
- Component-independant part of the adaptation framework.
  - **Coordinator.** The coordinator is responsible for choosing where the reaction is going to be executed within the execution path of the behavior. The possible locations are called candidate points; the chosen one is the adaptation point.
  - **Decider.** The decider decides when the component should adapt itself and chooses which reaction should be executed. To do so, it relies on the information given by the policy.

The coordinator and the decider both make decisions regarding the adaptation. However, they encapsulate separate concerns. The decider encapsulates the goal of the adaptation and the criterium that is going to be optimized; whereas the coordinator focuses on the mechanisms for enforcing the consistency of the dynamic adaptation.

4.2 Scenario of the Adaptation of a Component

From time to time, the decider decides that the component should adapt itself with one reaction. This may come from an external event (specified in the policy of the component); this may be a spontaneous decision (as a result of feedback control for example). Once the decision to adapt is made, the decider orders

the coordinator to execute the chosen reaction. Then, the coordinator chooses one candidate point in the execution path of the behavior. It also starts to monitor the execution of the behavior. When it reaches the adaptation point, the coordinator suspends the execution of the behavior and gives the execution control to the reaction. Once the reaction has finished, the behavior resumes its execution.

In addition, the behavior of a component can be a parallel code. In this case, the candidate points and the chosen adaptation point are global and represent global states. A global point is composed of one local point for each concurrent thread of the parallel behavior. Local points are identified by both their name in a model of the functional code and the indices in the iteration spaces.

The decider relies on the policy to make its decisions. The policy may contain an explicit set of rules: this can help the decider to choose the events to subscribe to. The policy may also contain performance models to help to choose the right reaction. Those performance models are parameterized with the content of the contracts describing the quality of the used services and resources.

5 Consistency of the Adaptation Within Parallel Components

When the component encapsulates a parallel code, the coordinator must choose a global point. However, the result of the execution of the adapted component must be semantically equivalent to the one of the component that does not adapt itself. Thus, the chosen point should be consistent with respect to a given property that garanties this semantic equivalence. The adaptation is said consistent if the reaction is executed from such a point.

A simple example illustrates this: if the component does not dead-lock in normal executions, it must not dead-lock when it adapts itself. Thus, choosing the point at which the reaction can be executed requires to be aware of the communications that occur in the behaviors and reactions of the component.

5.1 Consistency Model

The global point is R-consistent if and only if it satisfies the R relation. It is a n-ary relation if the parallel behavior is composed of n concurrent threads. This relation is defined over the n sets of local points. It is specific to each component.

In our example about dead-lock, this R relation encapsulates the required knowledge about communications within the component.

5.2 Classes of Parallel Components

Depending on the properties of the R relation, the parallel components can be classified. There is four major classes of parallel components.

SPMD components. If the R relation is *id* the identity, a global point is consistent if all the threads are locally at the same point. Regarding to the points, this means that all the threads share the same set of points. This case corresponds to the SPMD class of parallel applications.

Quasi-SPMD components. We consider that the R relation holds the following property:

$$\begin{aligned}&(R\,(p_1, p_2, \ldots, p_n) \wedge R\,(q_1, q_2, \ldots, q_n))\\ &\Rightarrow (p_1 \prec q_1 \wedge p_2 \prec q_2 \wedge \ldots \wedge p_n \prec q_n)\\ &\vee (q_1 \prec p_1 \wedge q_2 \prec p_2 \wedge \ldots \wedge q_n \prec p_n) \vee (q_1 = p_1 \wedge q_2 = p_2 \wedge \ldots \wedge q_n = p_n)\end{aligned}$$

The $\prec$ symbol denotes the strict "precede" relation over the sets of local points. In the execution paths, this is a total order relation.

If this property is satisfied, the local points can be renamed such that R becomes the identity relation. Thus, the component behaves like SPMD components.

Synchronous MPMD components. We suppose that the R relation satisfies the following property:

$$\begin{aligned}&(R\,(p_1, p_2, \ldots, p_n) \wedge R\,(q_1, q_2, \ldots, q_n))\\ &\Rightarrow (p_1 \preceq q_1 \wedge p_2 \preceq q_2 \wedge \ldots \wedge p_n \preceq q_n) \vee (q_1 \preceq p_1 \wedge q_2 \preceq p_2 \wedge \ldots \wedge q_n \preceq p_n)\end{aligned}$$

The $\preceq$ symbol denotes the reflexive "precede" relation over the sets of local points. In the execution paths, this is a total order relation.

In this case, the global points are still totally ordered in the execution path by a "precede" relation. This reflects the synchronization of the threads within the component. However, a local point can participate to several global points. Those components can be rewritten as quasi-SPMD components by duplicating such local points. This transformation restricts the consistency model.

Pessimistic parallel discrete event simulators belong to this class of components.

Asynchronous MPMD components. If none of the preceding properties is satisfied, the "precede" relation over global points in execution path is not a total order relation.

This class of components includes in particular master-slaves codes.

5.3 Comparison with Other Consistency Models

The model defined by PCL in [8] says that the adaptation is consistent if all the threads reach the i-th point (same i for all the threads) in the execution path. Although it seems similar, this is not equivalent to our *id*-consistency. Indeed, our model identifies points by name (identifier in the model of the code augmented with indices in iteration spaces) whereas PCL explicitly uses the rank of the point in the execution path.

Thus, our model accepts that the threads have different dynamic behaviors. The threads are not expected to execute the same number of iterations or to choose the same branch of conditional instructions. On the other hand, PCL supposes that all the threads remain in sync.

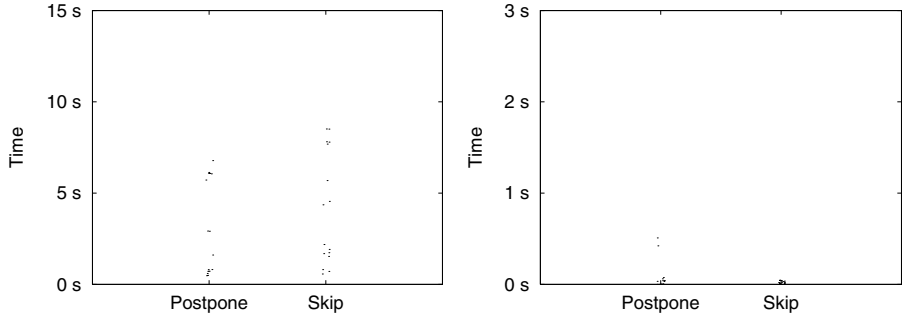
6 Realisation

By the time, we have designed and implemented an algorithmic solution for the coordinator. Our work restricts the model to the *id*-consistency in the case of SPMD parallel components. The candidate global points are exclusively looked for in the future of the execution path of the threads.

We have chosen to restrict to candidate points in the future. We worked around the impossibility of predicting the next point in the future of the execution path by introducing several strategies: the "postpone" strategy delays the prediction until the conditional instruction is executed; the "skip" strategy ignores the candidate points within the branches of the conditional instruction.

Figure 4 compares these two strategies. This experiment has been done with the NPB 3.1 [10] FFT code on a 4 PCs cluster running PadicoTM [11], which permits us to mix several middlewares such as MPI and CORBA within a single application. Only the coordinator has been implemented: the reaction is empty (no real adaptation); the decider is simulated by a trigger provided to the user. The encapsulation within a component has not been done as it has no influence for this experiment if we consider that the whole FFT code is within a single component.

Each dot of figure 4 represents one trigger of the adaptation. It appears on figure 4(a) that the "postpone" strategy generally chooses adaptation points that come sooner than the "skip" strategy. On the other hand, the figure 4(b) shows that the "postpone" strategy suspends more frequently the functional code



(a) In regard to the delay before adaptation

(b) In regard to time during which the functional code is suspended

Fig. 4. Comparison of the "postpone" and the "skip" strategies

than the "skip" strategy. Indeed, we observe that most of the experiments with the "skip" strategy causes no suspension (high density of dots at 0 s), whereas the functional code is frequently suspended with the "postpone" strategy (high density of dots up to about 0.5 s). These two observations exhibit the trade-off between the precision of the prediction and the risk of suspending the functional code.

7 Conclusion

In this paper we have shown that adaptation of parallel components can be achieved using our framework. Such an adaptation process needs to develop a consistency model and an algorithm to enforce this consistency before adaptation: the *id*-consistency model we developped shows the relevance of our consistency model for the adaptation of SPMD parallel codes.

We have shown through the FFT example that there is a trade-off between the precision of the choice of the point (best theoretical result) and the risk of uselessly suspending the execution of the functional code.

Futur works around the coordinator will consist in extending our implementation of the consistency model for non-SPMD components.

We plan to fully implement our framework, including a smart decider. This decider will be organized as a rule-based system. We will study how to represent usefull information for decision-making, such as states or changes of the environment. The representation of the adaptation policy have also to be defined. For this work we plan to rely on previous works on monitoring such as Delphoi [12] or SAJE/RAJE [13] and on our previous works on adaptation for mobile computing [14].

Another direction of our work is the study of the adaptation of assemblies of components for complete Grid applications. It is clear that in many applications, one component will not be able to adapt itself without taking into account the other components of the application. The role of the decider will be extended to include negotiation with deciders of other components of the application.

References

1. Aldinucci, M., Campa, S., Coppola, M., Danelutto, M., Laforenza, D., Puppin, D., Scarponi, L., Vanneschi, M., Zoccolo, C.: Components for high performance grid programming int the grid.it project. In: Workshop on Component Models andd Systems for Grid Applications. (2004)
2. Aldinucci, M., Coppola, M., Danelutto, M., Vanneschi, M., Zoccolo, C.: Assist as a research framework for high-performance grid programming environments. Technical Report TR-04-09, Università di Pisa, Dipartimento di Informatica, via F. Buonarroti 2, 56127 Pisa, Italy (2004)
3. Dongarra, J., Eijkhout, V.: Self-adapting numerical software for next generation application (2002)

4. Kennedy, K., Mazina, M., Mellor-Crummey, J., Cooper, K., Torczon, L., Berman, F., Chien, A., Dail, H., Sievert, O., Angulo, D., Foster, I., Gannon, D., Johnsson, L., Kesselman, C., Aydt, R., Reed, D., Dongarra, J., Vadhiyar, S., Wolski, R.: Toward a framework for preparing and executing adaptive grid programs. In: Proceedings of NSF Next Generation Systems Program Workshop (IPDPS). (2002)
5. Adve, V., Lam, V.V., Ensink, B.: Language and compiler support for adaptive distributed applications. In: ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah (2001)
6. Vadhiyar, S., Dongarra, J.: GrADSolve: RPC for high performance computing on the grid. In: Euro-Par 2003: Parallel Processing. Volume 2790. (2003)
7. Whaley, R.C., Petitet, A., Dongarra, J.J.: Automated empirical optimizations of software and the ATLAS project (2000)
8. Ensink, B., Adve, V.: Coordinating adaptations in distributed systems. In: 24th International Conference on Distributed Computing Systems. (2004) 446–455
9. Ségura-Devillechaise, M., Menaud, J.M., Muller, G., Lawall, J.: Web cache prefetching as an aspect: Towards a dynamic-weaving based solution. In: Proceedings of the 2nd international conference on Aspect-oriented software development, ACM Press (2003) 110–119
10. Nas parallel benchmark. (http://www.nas.nasa.gov/Software/NPB/)
11. PadicoTM. (http://www.irisa.fr/paris/Padicotm/welcome.htm)
12. Maassen, J., van Nieuwpoort, R.V., Kielmann, T., Verstoep, K.: Middleware adaptation with the delphoi service. In: AGridM 2004 - Proceedings of the 2004 Workshop on Adaptive Grid Middleware, Antibes Juan-Les-Pins, France (2004)
13. Guidec, F., Sommer, N.L.: Towards resource consumption accouting and control in java: a practical experience. In: Workshop on Resource Management for Safe Language, ECOOP 2002, Malaga, Spain (2002)
14. Chefrour, D., André, F.: Développement d'applications en environnements mobiles à l'aide du modèle de composant adaptatif aceel. In: Langages et Modèles à Objets LMO'03. Actes publiés dans la Revue STI. Volume 9 of L'objet. (2003)

Improving Multilevel Approach for Optimizing Collective Communications in Computational Grids

Boro Jakimovski and Marjan Gusev

University Sts. Cyril and Methodius,
Faculty of Natural Sciences and Mathematics,
Institute of Informatics,
Arhimedova 5, 1000 Skopje, Macedonia
{boroj, marjan}@ii.edu.mk

Abstract. Collective operations represent a tool for easy implementation of parallel algorithms in the message-passing parallel programming languages. Efficient implementation of these operations significantly improves the performance of the parallel algorithms, especially in the Grid systems. We introduce an improvement of multilevel algorithm that enables improvement of the performance of collective communication operations. An implementation of the algorithm is used for analyzing its characteristics and for comparing its performance it with the multilevel algorithm.

1 Introduction

Computational Grids [1] represent a technology that will enable ultimate computing power at the fingertips of users. Today Grids are evolving in their usability and diversity. New technologies and standards are used for improving their capabilities. Since this powerful resources need to be utilized very efficiently we need to adopt the programming models used in the parallel and distributed computing. One of the main problems facing parallel and distributed computing when introduced to the Grid environment is scalability.

Currently most widely used message passing parallel programming standard is the MPI standard [2]. MPI represents a programming standard that enables implementation of parallelism using message passing. Operations for Collective communication represent a part of the MPI standard that involves communications between a group of processes. Optimizations of collective communications have been the focus of many years of research. This research has led to development of many different algorithms for implementation of collective communications [3]. These algorithms were optimized mainly for cluster computations where the characteristics of the communications between every two nodes are the same.

Main problem of introducing MPI to the Grid environment is the big latency of the communications. Even bigger problem lies in the different latencies of different pairs of processes involved in the communication. This led to the development of new improved algorithms for implementation of collective communication in the Grid environment. Most algorithms for implementation of collective communications are

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 548 – 556, 2005.

based on tree like communication pattern. There have been many efforts for optimization of the topology of this communication trees for better performance in the Computational Grids. In this paper we introduce an improvement of the multilevel approach for optimization of collective communications, using an adaptive tree algorithm called α tree.

In Section 2, we give a brief overview to the previous solutions for optimization of collective communications in Computational Grids and describe the multilevel approach for optimization of collective communications. In Section 3, we introduce the improvement of the multilevel approach called Multilevel communication α tree. In Section 4, we present the results of the experiments for the evaluation of the newly proposed algorithm. Finally in Section 5, we give a brief conclusion and the direction for future research.

2 Topology Aware Collective Communications

There have been different approaches for solving the problem of optimizing communication tree for collective operations in Computational Grids. First efforts started with the development of algorithms that involved Minimal Spanning Tree [4], followed by variations of this approach by changing the weights and conditions in the steps for building the communication tree (SPOC [5], FNF [5], FEF [6], ECEF [6], Look-ahead [6], TTCC [7]).

Currently best performing solution is the solution utilizing the network topology information for building the communication tree. This approach, later called topology aware collective communication, was introduced in [4] and later improved in [8] and [9]. This algorithm involved grouping of the processors in groups where each group represent either processors from one multiprocessor computer or processors from one cluster. Once the groups are defined, the communication tree is defined in two levels. The first level contains one group consisting of the root processes from each group. The second level contains the groups defined previously.

The main disadvantage of the two-level algorithm was the utilization of only two levels of communication, local area communication (low latency) and wide area communication (high latency). This disadvantage was overthrown by implementation of multilevel communication pattern introduced by Karonis et. all in [10]. Their approach, implemented in MPICH-G2 [11], defines up to four levels of network communication. Each level consists of several groups of processors where the communications have common characteristics. This way they achieve more adequate topology aware communication pattern which is very close to the optimal.

3 Multilevel Communication α Tree

The multilevel communication tree improves the communication time of collective communications by introducing better topology awareness. The only disadvantage of multilevel communication tree is the choice of communication algorithms. Authors settle for simple solution where they choose one algorithm for high latency level (the first level – wide area level) and another algorithm for low latency levels (all other

levels – local area and intra machine). The algorithm chosen for high latency communications is flat tree, which has been shown to behave optimally in such conditions. For low latency communications, the authors choose binomial tree, which also has been shown by LogP model [12] to be optimal in such conditions.

3.1 α Communication Tree

One of the most advanced algorithms for implementation of collective communication operations is the α-tree algorithm [13]. The algorithm is derived from the theoretically optimal algorithm of λ-tree. The α-tree algorithm overcomes the problems for implementation of the λ-tree but with reduced optimality.

The α-tree algorithm represents a communication tree dependent of the value of the parameter α. The value of the parameter is in the range between 0 and 0.5. The main characteristic of the α-tree is that for α=0, the α-tree looks like flat tree, and for α=0.5 the α-tree looks like binomial tree. This characteristic shows that the α-tree algorithm can adjust itself according to the network characteristics, i.e. if the latency of the communications is low then the value of the parameter will shift towards 0.5, and if the latency of the communications is high then the value of the parameter will move towards 0. This behavior is visually represented in Fig. 1.

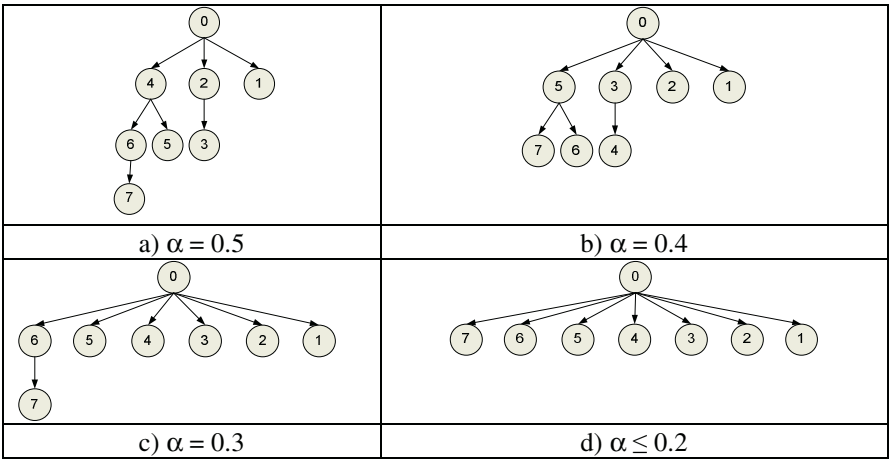


Fig. 1. Tree topology changes from binomial tree a) to flat tree d) as the value of the parameter α changes from 0.5 to 0

3.2 Multilevel Communication α Tree

The disadvantage of the Multilevel communication tree algorithm lies in the choice of the algorithms for communication inside the groups of processes on each level. Our approach tries to improve this disadvantage by defining new communication scheme which will be more efficient then the multilevel tree.

Multilevel communication α tree represents an improvement of the multilevel algorithm. The improvement lies in the ability to properly choose the communication algorithm for each of the four levels of the multilevel communication tree algorithm. The best way to implement the communication algorithm for each group of the communication tree is to properly adjust the communication algorithm according to the latency characteristics of the network. We choose to use the α-tree algorithm for each level of the multilevel tree but with different values of the parameter α for each group. This should enable more flexibility for the algorithm, enabling it to achieve better performance. As it can be seen from the characteristics of the multilevel α tree algorithm, the multilevel tree algorithm represents a special case of the new algorithm when we choose value of $\alpha=0$ for the first level and value of $\alpha=0.5$ for all of the other levels. Therefore we can conclude that the new algorithm should enable better performance then the multilevel algorithm.

4 Experimental Results

4.1 Simulation Environment

The evaluation of the proposed solution for implementation of collective communication was conduced in our relatively small Grid infrastructure. For evaluation purposes we have changed the topology of our Grid by making it more suitable to achieve real results from the simulation. Our simulation resources were four laboratories each with 20 PCs installed with Red Hat Linux and Globus 2.4. The laboratories are separated in two buildings connected between with a link, which we have reduced to 10 Mbit link. Each laboratory we further separated in several clusters of 4-5 PCs. The overall Grid infrastructure used for evaluation of the new algorithm is depicted in Fig.2.

From the figure it can be seen that the communication infrastructure on the second and third level is 100 Mbit switch/hub. To simulate different network scenarios we made different experiments in the grid infrastructure either by using switch technology, or by using hub technology.

4.2 Measurement Technique

Since the measurement of the performance is the crucial part of the evaluation process it is very important to choose the correct measuring technique. Measuring collective communication requires measurement of consecutive execution of many operation calls since one operation call is very short and cannot be measured correctly. Main measurement problem is the pipelining effect that is generated by consecutive calls of the collective operation [14][15].

The pipelining effect can easily be solved by introducing barriers between consecutive calls of the operations. This approach cannot be implemented straight forward because of the problems that arise with the barrier implementation. When using already implemented barrier from the MPICH library (topology unaware), the main problem lies in the very slow solution for the barrier. This slow solution cannot be used since if the barrier execution time varies only by few percents then the results

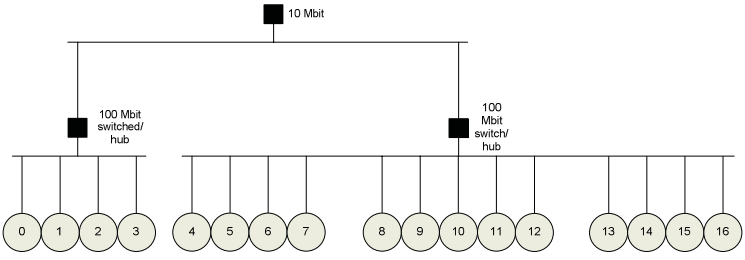


Fig. 2. Topology of the Grid infrastructure used for the experiments for evaluating the new algorithm

from the collective operation will be distorted completely. In such case the use of some specially tailored barrier is needed. This barrier will not be a real barrier but a pseudo barrier that in these circumstances will ensure that the execution will not be pipelined.

Our choice for semi-barrier is the use of opposite ring topology for communication. This means that the between each collective communication operation the processes synchronize between each other by communicating in a ring where each process (with rank r) receives a message from the process with rank $r + 1$ and once it receives the message it resends it to the process with tank $r - 1$. This process starts with the root process and ends with it. This solution represents simple, efficient barrier that reduces pipelining effect in the operation and the barrier.

4.3 Experimental Results

Performance measurement of the new algorithm is evaluated by using the broadcast operation. In order to fully evaluate the new algorithm we have measured the performance using many different scenarios. To achieve this in our experiments we have changed the following parameters:

- message size
- parameter α for the second level
- parameter α for the third level
- characteristics of the network topology

The parameter α for the first level is fixed to the value of 0 because this level contains only two processes and any value of α will lead to the same communication topology.

The results of the experiments are depicted in the figures of this chapter. The figures represent three dimensional charts where on the two axes (x and y) are represented the α parameters for the second and the third level of the communication tree. The third axes (z – depth) represents the measured time of the operation for the particular values of α for the second and third level. The shading of the charts represents the value of the z-axes for easier reading. If the shade is darker then the time is lower.

On Fig. 3 we can see the results for the measurements for different network characteristic. The figure shows how the performance changes once the network latency increments. The first part of the figure shown the performance of the hub infrastructure. This infrastructure characterizes with bigger latency because of the non-parallel communications. This makes the binomial tree very inefficient because of its low latency and parallel nature. This shows that the optimum shifts towards the lower values of the parameters α. On the other hand the for the switch infrastructure the optimum moves toward the higher values for α, but still doesn't achieve value of 0.5 for both levels which is the case of the multilevel algorithm.

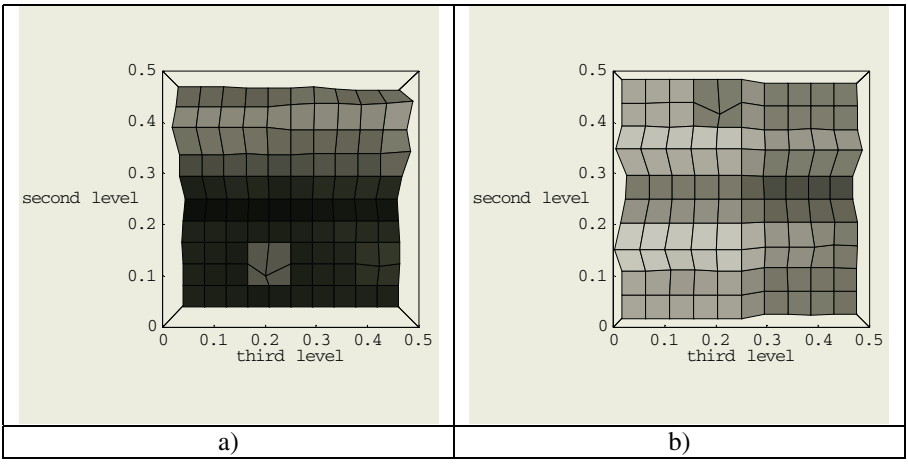


Fig. 3. Results from the simulations using different network topology: a) hub infrastructure, b) switch infrastructure

The second analyzed aspect during the simulations was the effect of the message size over the performance of the operations. Results gathered from the simulations are shown on Fig. 4. As it can be seen from the figures the optimum for the values of the parameter α change as the message size grows. The conducted experiments were for message sizes from 1 byte to 16 KB with the step "power of 2". The results shown on the picture are from 2 KB and above since the results less then 2 KB are identical to the 2 KB results. Reason for this is the usage of TCP/IP protocol that sends messages of approximately 1KB in size even if we send messages with 1 byte of data. The results show the desired performance improvement. It can be concluded that for smaller packet sizes the optimum tends to move towards the lower values of parameters α, and for bigger packet size the optimum moves towards the higher values of the parameters. This is expected since the usage of TCP/IP as transport protocol. When TCP/IP segments large packets into small packets the network communication is flooded with packets and in such case the parallelism in communication is increased.

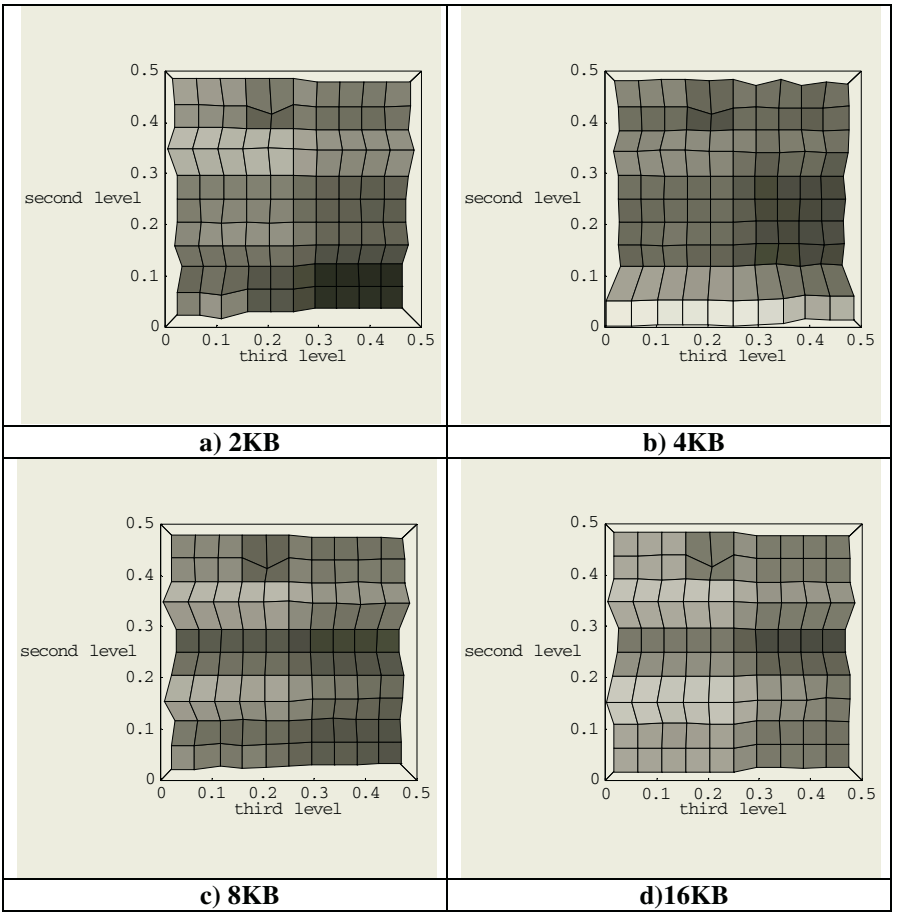


Fig. 4. Results from the simulations using different message size

4.4 Performance Improvement

As it could be expected from the analytical characteristics, experimental results show that the new algorithm gives the opportunity for improvement of the performance of collective communications. The overall improvements gathered from the experimental results are shown on Table 1 and Table 2. The results shown represent the optimal time for the given α parameters using the multilevel α tree algorithm (Optimal time). For comparison with the multilevel algorithm, we give the measured time of the multilevel algorithm (Multilevel time). The results show that in switch topology the improvements are around 15% compared to the multilevel algorithm. In hub infrastructure the improvements are smaller for the bigger data size, since this introduces many packets to the network which makes even flat tree behave as parallel algorithm, but for smaller data size the improvements are significant and achieve 40% improvement.

Table 1. Performance improvement for switch topology

Packet size	Optimal time	Multilevel time	α second	α third	Improvement
16KB	42.5472	48.5963	0.3	0.3	12.45%
8KB	24.4484	26.3964	0.25	0.35	7.38%
4KB	9.82545	12.498	0.1	0.35	21.38%
2KB	5.76205	7.16053	0.1	0.4	19.53%
1KB	3.64349	4.04574	0.1	0.35	9.94%
512B	3.17763	3.68794	0.1	0.45	13.84%

Table 2. Performance improvement for hub topology

Packet size	Optimal time	Multilevel time	α second	α third	Improvement
16KB	116.988	125.192	0.25	0	6.55%
8KB	61.2282	64.0655	0.25	0	4.43%
4KB	33.6797	34.9603	0.5	0.25	3.66%
2KB	18.0641	18.6563	0.25	0.25	3.17%
1KB	10.1231	18.4963	0	0.1	45.27%
512B	5.59515	9.34781	0	0.15	40.14%

5 Conclusion

As Computational Grids are more widely used, the need for new techniques for parallel and distributed computing are needed. Topology aware communications are essential aspect of improvement of parallel programs. The currently adopted multilevel tree topology aware solution achieves great performance improvements. Still the simple solution in algorithm performance limits the ability to fully utilize the topology and network characteristics.

Our approach represents an improvement to the multilevel approach for optimizing collective communications in Computational Grids. The usage of the α tree algorithm for adopting the different network characteristics enables significant improvement in the collective operations performance. The new algorithm doesn't increase the implementation issues since the α-tree algorithm is very easy to implement and is easily fitted in the MPICH-G2 implementation of topology aware collective communications that utilize communication trees. The only problem is the choice of the parameter α, and its distribution during communicator creation.

Simulation of the new algorithm shows the possibility for improving the performance of collective communications in Computational Grids. Because the measurement of the performance of the algorithm is a very important issue, we gave special attention on problems concerning the measurement techniques. Our choice of measurement technique is consecutive call of the collective operation followed by a semi-barrier implemented using opposite ring communication topology.

Future work is research in developing an efficient and easy way of utilizing this new approach for use in Computational Grids. This will require a mechanism for gathering and using network characteristics for automatic parameter selection.

References

1. Foster, I., Kesselman, C. ed.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1999.
2. Message Passing Interface Forum: MPI: A message-passing interface standard. International Journal of Supercomputer Applications, 8(3/4) (1994) 165-414
3. Vadhiyar, S. S., Fagg, G. E., Dongarra, J.: Automatically Tuned Collective Communications. Proceedings of the IEEE/ACM SC2000 Conference, Dallas, Texas (2000)
4. Lowekamp, B. B., Beguelin. A.: ECO: Efficient Collective Operations for communication on heterogeneous networks. Proc. of 10th Intl. Parallel Processing Symposium, (1996) 399–405
5. Banikazemi, M., Moorthy, V., Panda, D.: Efficient Collective Communication on Heterogeneous Networks of Workstations. International Conference on Parallel Processing. Minneapolis, MN (1998) 460–467
6. Bhat, P.B., Raghavendra, C.S., Prasanna, V.K.: Efficient Collective Communication in Distributed Heterogeneous Systems. Proceedings of the International Conference on Distributed Computing Systems (1999)
7. Cha, K., Han, D., Yu, C., Byeon, O.: Two-Tree Collective Communication in Distributed Heterogeneous Systems. IASTED International Conference on Networks, Parallel and Distributed Processing, and Applications (2002)
8. Kielmann, T., Hofman, R. F. H., Bal, H. E., Plaat, A., Bhoedjang, R. A. F.: MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems. Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP), Atlanta, GA, (1999) 131–140
9. Kielmann, T., Bal, H. E., Gorlatch, S.: Bandwidth-efficient Collective Communication for Clustered Wide Area Systems. IPDPS 2000, Cancun, Mexico (2000)
10. Karonis, N., de Supinski, B., Foster, I., Gropp, W., Lusk, E., Bresnahan, J.: Exploiting hierarchy in parallel computer networks to optimize collective operation performance. Proc. of the 14th International Parallel and Distributed Processing Symposium, (2000) 377–384
11. MPICH-G2 web page. http://www.globus.org/mpi.
12. Culler, D.E., Karp, R., Patterson, D.A., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T: LogP: Towards a realistic model of parallel computation. Proceedings of the 4th SIGPLAN Symposium on Principles and Practices of Parallel Programming, (1993) 1–12
13. Bernaschi, M., Iannello, G.: Collective Communication Operations: Experimental Results vs. Theory. Concurrency: Practice and Experience, (1998), 10(5):359–386
14. de Supinski, B., Karonis, N.: Accurately Measuring MPI Broadcasts in a Computational Grid. In The Eighth IEEE International Symposium on High Performance Distributed Computing. IEEE Computer Society Press, (1999).
15. Lacour, S.: MPICH-G2 collective operations: performance evaluation, optimizations. Rapport de stage MIM2, Magistère d'informatique et modélisation (MIM), ENS Lyon, MCS Division, Argonne National Laboratory, USA, (2001)

Rough Set Based Computation Times Estimation on Knowledge Grid

Kun Gao[1,2], Youquan Ji[3], Meiqun Liu[4], and Jiaxun Chen[1]

[1] Information Science and Technology College, Donghua University, P.R.C
[2] Aviation University of Air Force, P.R.C
[3] Geo-Exploration Science and Technology College, Jilin University, P.R.C
[4] Administration of Radio Film and Television of Jilin Province, P.R.C
gaokun@mail.dhu.edu.cn

Abstract. Efficient estimating the application computation times of data mining is a key component of successful scheduling on Knowledge Grid. In this paper, we present a holistic approach to estimation that uses rough sets theory to determine a reduct and then compute a runtime estimate. The heuristic reduct algorithm is based on frequencies of attributes appeared in discernibility matrix. We also present to add dynamic information about the performances of various data mining tools over specific data sources to the Knowledge Grid service for supporting the estimation. This information can be added as additional metadata stored in Knowledge Metadata Repository of Grid. Experimental result validates our solution that rough sets provide a formal framework for the problem of application run times estimation in Grid environment.

1 Introduction

Knowledge Grid is a software architecture for geographically distributed PDKD (Parallel and Distributed Knowledge Discovery) systems [1]. This architecture is built on top of a computational Grid and Data Grid that provides dependable, consistent, and pervasive access to high-end computational resources[2][3]. The Knowledge Grid uses the basic Grid services and defines a set of additional layers to implement the services of distributed knowledge discovery on world wide connected computers where each node can be a sequential or a parallel machine.

A key aspect of scheduling data mining applications on Knowledge Grid is the ability to accurately estimate the computation times. Such techniques can improve the performance of scheduling algorithms and help estimate queue times in Knowledge Grid environments. For example, the Knowledge Grid provides a specialized broker of Grid resources for PDKD computations: given a user's request for performing a data mining analysis, the broker takes allocation and scheduling decisions, and builds the execution plan, establishing the sequence of actions that have to be performed in order to prepare execution (e.g., resource allocation, data and code deployment), actually execute the task, and return the results to the user. The execution plan has to satisfy given requirements (such as performance, response time, and mining algorithm) and

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 557 – 566, 2005.

constraints (such as data locations, available computing power, storage size, memory, network bandwidth and latency). Once the execution plan is built, it is passed to the Grid Resource Management service for execution. Clearly, many different execution plans can be devised, and the Resource Allocation and Execution Management services have to choose the one which minimizes response time. In its decision making process, this service has to accurately estimate applications processing or computation times so as to help estimate queue times and improve the performance of scheduling algorithms. Unfortunately, the computation times depend on many factors: data size, specific mining parameters provided by users and actual status of the Grid etc. Moreover, the correlations between the items present in the various transactions of a dataset largely influence the response times of data mining applications. Thus, predicting its performance becomes very difficult.

Our application runtime prediction algorithms operate on the principle that applications with similar characteristics have similar runtimes. Thus, we maintain a history of applications that have executed along with their respective runtimes. To estimate a given application's runtime, we identify similar applications in the history and then compute a statistical estimate of their runtimes. We use this as the predicted runtime. The fundamental problem with this approach is the definition of similarity; diverse views exist on the criteria that make two applications similar. For instance, we can say that two applications are similar because the same user on the same machine submitted them or because they have the same application name and are required to operate on the same size data. Thus, we must develop techniques that can effectively identify similar applications. Such techniques must be able to accurately choose applications' attributes that best determine similarity. Having identified a similarity template, the next step is to estimate the applications' runtime based on previous, similar applications. We can use several statistical measures to compute the prediction, including measures of central tendency such as the mean and linear regression.

Due to the lack of centralized control and the dynamic nature of resource availability, we propose to include in the KDS (Knowledge Directory service) [1] dynamic information about the performances of the various data mining tools over specific data sources. This information can be added as additional metadata associated with datasets. Therefore, the extended KDS not only includes static metadata utilized to search data or components, but also dynamic information used for estimating the computation times. The dynamic metadata regards monitoring information about previous runs of the various software components on particular datasets.

In this paper, we present a holistic approach to estimation that uses rough sets theory to determine a similarity template and then compute a runtime estimate using identified similar applications. We tested the technique in a similar Grid environment. The rest of this paper is organised as follows: in section 2, we introduce related works and point out the limitations of some previous works. In section 3, we discuss the suitability of rough set to predict applications run times. In section 4, we recall necessary rough set notions used in the paper, and then present our reduct algorithm and application runtime estimation algorithm. In section 5, we conduct experiments to evaluate our approach. Finally section 6 concludes this paper.

2 Related Works

Early work in the parallel computing area proposed using similarity templates of application characteristics to identify similar tasks in a history. A similarity template is a set of attributes that we use to compare applications in order to determine if they are similar. Thus, for histories recorded from parallel computer workloads, one set of researchers selected the queue name as the characteristic to determine similarity [4]. They considered that applications assigned to the same queue were similar. In other work [5], researchers used several templates for the same history, including user, application name, number of nodes, and age.

Manually selecting similarity templates had the following limitations:

- Identifying the characteristics that best determine similarity isn't always possible.
- It's not generic: although a particular set of characteristics might be appropriate for one domain, it's not always applicable to other domains.

In [6][7], they proposed automated definition and search for templates and used genetic algorithms and greedy search techniques. They were able to obtain improved prediction accuracy using these techniques.

Recently, another effective approach to predict execution times on Grid is [8]. They investigate a use of sampling: in order to forecast the actual execution times of a given data mining algorithm on the whole dataset, they run the same algorithm on a small sample of the dataset. Many data mining algorithms demonstrate optimal scalability with respect to the size of the processed dataset, thus making the performance estimate possible and accurate enough. However, in order to derive an accurate performance model for a given algorithm, it should be important to perform an *off-line* training of the model, for different dataset characteristics and different parameter sets.

In this paper, we develop a rough sets based technique to address the problem of *automatically* selecting characteristics that best define similarity. In contrast to [6], our method determines a reduct as template, instead of using greedy and genetic algorithms. Rough sets provide an intuitively appropriate theory for identifying templates. The entire process of identifying similarity templates and matching tasks to similar tasks is based on rough sets theory, thereby providing an appropriate solution with a strong mathematical underpinning.

3 Rough Set Based Computation Times Estimation

Rough sets theory as a mathematical tool to deal with uncertainty in data provides us with a sound theoretical basis to determine the properties that define similarity. Rough sets operate entirely on the basis of the data that is available in the history and require no external additional information. The history represents an information system in which the objects are the previous applications whose runtimes and other properties have been recorded. The attributes in the information system are these applications' properties. The decision attribute is the application runtime, and the other recorded properties constitute the condition attributes. This history model intuitively facilitates reasoning about the recorded properties so as to identify the dependency between the recorded attributes and the runtime. So, we can concretize similarity in terms of the condition attributes that are relevant and significant in determining the runtime. Thus,

the set of attributes that have a strong dependency relation with the runtime can form a good similarity template. Having cast the problem of application runtime as a rough information system, we can examine the fundamental concepts that are applicable in determining the similarity template.

The objective of similarity templates in application runtime estimation is to identify a set of characteristics on the basis of which we can compare applications. We could try identical matching, i.e. if *n* characteristics are recorded in the history, two applications are similar if they are identical with respect to all n properties. However, this considerably limits our ability to find similar applications because not all recorded properties are necessarily relevant in determining the runtime. Such an approach could also lead to errors, as applications that have important similarities might be considered dissimilar even if they differed in a characteristic that had little bearing on the runtime.

A similarity template should consist of the most important set of attributes that determine the runtime without any superfluous attributes. A reduct consists of the minimal set of condition attributes that have the same discerning power as the entire information system. In other words, the similarity template is equivalent to a reduct that includes the most significant attributes. Finding a reduct is similar to feature selection problem. All reducts of a dataset can be found by constructing a kind of discernibility function from the dataset and simplifying it. Unfortunately, it has been shown that finding minimal reduct or all reducts are both NP-hard problems. Some heuristics algorithms have been proposed. Hu gave an algorithm using significant of attribute as heuristics [9]. Starzyk used strong equivalence to simplify discernibility function [10]. Some algorithms using genetic algorithm have been also proposed. However, there are no universal solutions. It's still an open problem in rough set theory.

Rough sets theory has highly suitable and appropriate constructs for identifying the properties that best define similarity for estimating application runtime. A similarity template must include attributes that significantly affect the runtime and eliminate those that don't. This ensures that the criteria with which we compare applications for similarity have a significant bearing on determining runtime. Consequently, applications that have the same characteristics with respect to these criteria will have similar runtimes.

In this paper, we propose a simple but useful heuristic reduct algorithm using discernibility matrix. The algorithm is based on frequencies of attributes appeared in discernibility matrix.

4 Heuristic Reduct Algorithm and Application Runtime Estimation Algorithm

In this section, we first recall necessary rough set notions [11] used in this section, and then present the reduct algorithm and application runtime estimation algorithm.

4.1 Related Rough Set Concepts

Definition 1 (information system)**.** An information system is an ordered pair S=(U, $A \cup \{d\}$), where U is a non-empty, finite set called the universe, A is a non-empty,

finite set of conditional attributes, d is a decision attribute. $A \cap \{d\} = \Phi$. The elements of the universe are called objects or instances.

Information system contains knowledge about a set of objects in term of a predefined set of attributes. The set of objects is called concept in rough set theory. In order to represent or approximate these concepts, an equivalence relation is defined. The equivalence classes of the equivalence relation, which are the minimal blocks of the information system, can be used to approximate these concepts. Concept can be constructed from these blocks are called definable sets. As to undefinable sets, two definable sets, upper-approximation set and lower-approximation set are constructed to approximate the concept.

Definition 2 (Indiscernibility relation)**.** Let $S=(U,A \cup \{d\})$ be an information system, every subset $B \subseteq A$ defines an equivalence relation $IND(B)$,called an indiscernibility relation,defined as $IND(B)=\{(x,y) \in U \times U: a(x)=a(y)$ for every $a \in B\}$.

Definition 3 (Positive region)**.** Given an information system $S= (U,A \cup \{d\})$, let $X \subseteq U$ be a set of objects and $B \subseteq A$ a selected set of attributes. The lower approximation of X with respect to B is $B_*(X)=\{x \in U:[x]_B \subseteq X\}$. The upper approximation of X with respect to B is $B^*(X)=\{x \in U: [x]_B \cap X \neq \Phi\}$. The positive region of decision d with respect to B is $POS_B(d)= \cup \{B_*(X): X \in U/IND(d)\}$

The positive region of decision attribute with respect to B represents approximate quantity of B. Not all attributes are necessary while preserving the approximate quantity of the original information system. Reduct is the minimal set of attribute preserving approximate quantity.

Definition 4 (Reduct)**.** An attribute a is dispensable in $B \subseteq A$ if $POS_B(d)= POS_{B-\{a\}}(d)$. A reduct of B is a set of attributes $B' \subseteq B$ such that all attributes $a \in B-B'$ are dispensable, and $POS_B(d)= POS_{B'}(d)$.

There are usually many reducts in an information system. In fact, one can show that the number of reducts of an information system may be up to $C^{|A|/2}_{|A|}$. In order to find reducts, discernibility matrix and discernibility function are introduced.

Definition 5 (discernibility matrix)**.** The discernibility matrix of an information system is a symmetric $|U| \times |U|$ matrix with entries c_{ij} defined as $\{a \in A | a(x_i) \neq a(x_j)\}$ if $d(x_i) \neq d(x_j)$, Φ otherwise. A discernibility function can be constructed from discernibility matrix by or-ing all attributes in c_{ij} and then and-ing all of them together. After simplifying the discernibility function using absorption law, the set of all prime implicants determines the set of all reducts of the information system.

4.2 Heuristic Reduct Algorithm

The heuristic comes from the fact that intersection of a reduct and every items of discernibility matrix can not be empty. If there are any empty intersections between some item c_{ij} with some reduct, object i and object j would be indiscernible to the reduct. And this contradicts the definition that reduct is the minimal attribute set discerning all objects (assuming the dataset is consistent).

A straightforward algorithm can be constructed based on the heuristic. Let candidate reduct set $R=\Phi$. We examine every entry c_{ij} of discernibility matrix. If their intersection is empty, a random attribute from c_{ij} is picked and inserted in R; skip the entry otherwise. Repeat the procedure until all entries of discernibility matrix are examined. We get the reduct in R.

The algorithm is simple and straightforward. However, in most times what we get is not reduct itself but superset of reduct. For example, there are three entries in the matrix: $\{a_1, a_3\}$, $\{a_2, a_3\}$, $\{a_3\}$. According the algorithm, we get the reduct $\{a_1, a_2, a_3\}$ although it is obvious $\{a_3\}$ is the only reduct. This is because our heuristic is a necessary but not sufficient condition for a reduct. The reduct must be a minimal one. The above algorithm does not consider this. In order to find reduct, especially shorter reduct in most times, we need more heuristics.

A simple yet powerful method is to sort the discernibility matrix according to $|c_{ij}|$. As we know, if there is only one element in c_{ij}, it must be a member of the reduct. We can image that attributes in shorter and frequent $|c_{ij}|$ contribute more classification power to the reduct. After sorting, we can first pick up more powerful attributes, avoid situations like in the example mentioned above, and more likely get optimal or sub-optimal reduct.

The sort procedure is like this. First, all the same entries in the discernibility matrix are merged and their frequency is recorded. Then the matrix is sorted according to the length of every entry. If two entries have the same length, more frequent entry takes precedence.

Input: an information system $(U, A \cup \{d\})$, where $A= \cup a_i$, $i=1,\ldots,n$.
Output: a reduct *Red*

1. $Red=\Phi$, count (a_i) =0, for $i=1,\ldots n$.
2. Generate discernibility matrix M and count frequency of every attribute count(a_i);
3. Merge and sort discernibility matrix M;
4. For every entry m in M do
5. If $(m \cap Red = = \Phi)$
6. select attribute a with maximal $count(a)$ in m
7. $Red=Red \cup \{a\}$
8. Endif
9. EndFor
10. Return *Red*

Fig. 1. A Heuristic Reduct Algorithm

When generating the discernibility matrix, the frequency of every individual attribute is also counted for later use. The frequencies are used in helping picking up attribute when it is needed to pick up one attribute from some entry to insert into the reduct. The idea is that a more frequent attribute is more likely to be a member of the reduct. The counting process is weighted. Similarly, attributes appeared in shorter entry get higher weight. When a new entry c is computed, the frequency of the corresponding attribute $f(a)$ is updated as $f(a)=f(a)+|A|/|c|$, for every $a \in c$; where $|A|$ is

total attribute of information system. For example, let $f(a_1) = 3$, $f(a_3) = 4$, the system have 10 attributes in total, and the new entry is $\{a_1, a_3\}$. Then frequencies after this entry can be computed: $f(a_1)=3+10/2=8$; $f(a_3)=4+10/2=9$.

Fig. 1 is a heuristic reduct algorithm written in pseudo-code. In line 2, when a new entry c of M is computed, $count(a_i)$ is updated. $count(a_i):=count(a_i)+n/|c|$ for every $a_i \in |c|$. In line 3, the same entries are merged and M is sorted according to the length and frequency of every entry. Line 4-9 traverses M and generates the reduct.

4.3 Application Runtime Estimation Algorithm

Let's now look at the estimation algorithm as a whole. Its input is a history record of application characteristics collected over time, specifically including actual recorded runtimes, and a task T with known parameters whose runtime we wish to estimate.

Step 1. Partition the history into decision and condition attributes. The recorded runtime is the decision attribute, and the other recorded characteristics are the condition attributes. The approach is to record a comprehensive history of all possible statistics with respect to an application because identifying the attributes that determine the runtime isn't always possible.

Step 2. Apply the rough sets algorithm to the history and identify the similarity template.

Step 3. Combine the current task T with the history H to form a current history HT.

Step 4. Determine from HT the equivalence classes with respect to the identified similarity templates. This implies grouping into classes previous tasks in the history that are identical with respect to the similarity template. Because the similarity template generated using rough sets is a reduct, this leads to the equivalence classes consisting of previous tasks that are identical with respect to the characteristics that have the most significant bearing on the runtime. In this case, rough sets provide a basis for identifying the similarity template and finding previous tasks that match the current task by the intuitive use of equivalence classes. Thus, we integrate the process of matching the current task with previous tasks in the history into the overall process of estimating application runtime.

Step 5. Identify the equivalence class EQ to which T belongs.

Step 6. Compute the mean of the runtimes of the objects: $EQ \cap H$.

Input: History of tasks=H, Current task for which the runtime has to be estimated=T.

Output: Estimated runtime EST.

1. Partition H such that the runtime is the decision attribute and all the other recorded characteristics are the condition attributes.
2. Apply the rough sets algorithm to the history and generate a similarity template ST.
3. Let $H_T = H + T$, where H and T are union compatible.
4. Compute equivalence classes of H_T with respect to ST.
5. Identify the equivalence class EQ_T to which T belongs.
6. Compute the mean of the recorded runtimes EST in H for all objects in EQ_T.

Fig. 2. Application Runtime Estimation Algorithm

The formal algorithm Fig. 2 offers a formal view of the estimation algorithm. As we explained previously, the entire process of identifying similarity templates and matching tasks to similar tasks is based on rough sets theory, thereby providing an appropriate solution with a strong mathematical underpinning.

5 Experiments and Results

This experiment aims to validate the prediction accuracy of our rough sets algorithm and investigate the impact of varying the number of condition attributes on prediction performance. We applied our approach in estimating the computation times of data mining tasks. We differentiated the test case from the historical records by removing the runtime information and varying the number of attributes. Thus, all test cases consist of attributes specified except the recorded runtime. The runtime included data transmission and algorithm execution. The idea was to determine an estimated runtime using our prediction technique and compare it with the task's actual runtime.

We compiled a history of data mining tasks by running several data mining algorithms and recording information about the tasks and environment. We executed several runs of data mining jobs by varying the jobs' parameters such as the mining algorithm, the data sets, and the sizes of the data sets and so on. Several data sets with sizes varying from 1 to 20 Mbytes were generated.

Table 1. Condition Attributes and Corresponding Reduct in Each Experiment

Experiment Number	Condition Attributes	Reduct
1	time, algorithm, parameter, disk cache, data size	algorithm, parameter, data size
2	operating system, time, algorithm, parameter, disk cache, data size, dimensionality, file name	algorithm, parameter, data size, dimensionality
3	CPU type, operating system, time, algorithm, parameter, disk cache, data size, dimensionality, file name, operating system version, CPU speed	algorithm, parameter, data size, dimensionality, CPU speed
4	memory type, CPU type, operating system, time, algorithm, parameter, disk cache, data size, dimensionality, file name, operating system version, CPU speed, available memory, disk type	algorithm, parameter, data size, dimensionality, CPU speed, available memory
5	IP, memory type, CPU type, operating system, time, algorithm, parameter, disk cache, data size, dimensionality, file name, operating system version, CPU speed, available memory, disk type, bandwidth, mainboard bus	algorithm, parameter, data size, dimensionality, CPU speed, available memory, bandwidth

The simulated environment is similar to an actual Grid environment. It is composed of fifteen machines installed with GT3. Those machines have different physical configurations (CPU, memory, disk, and network adaptor etc.), operating systems (windows 2000 and Linux) and bandwidth of network. We used histories with 100 and 150 records and each experimental run consisted of 25 tests. Table 1 shows the condition attributes and corresponding reduct in each experiment. Fig. 3 illustrates the impact of prediction performance by varying the number of condition attributes.

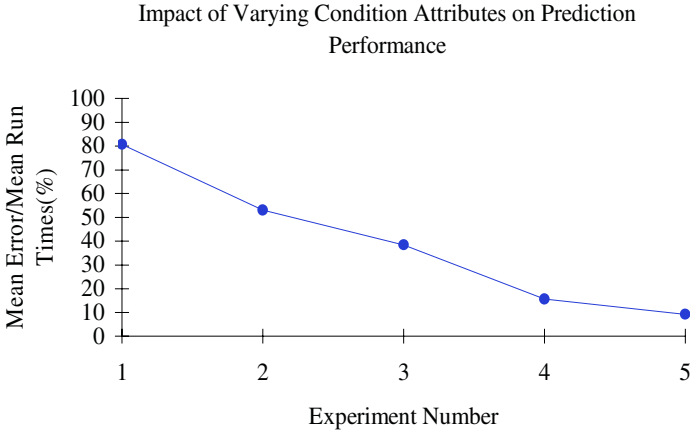


Fig. 3. The Number of Condition Attributes Vs. Prediction Performance

The measure used for comparison is the percentage of the mean error to the mean run times. A lower value indicates the better prediction accuracy. The result we recorded was 9.28%. It is very low, which indicates that we obtained very good estimation accuracy for data mining tasks. Because rough sets operate entirely on the basis of the condition attributes available in the history and require no external additional information, thus the more abundant the information correlating with performance, the more accurate the prediction is.

6 Conclusion

We have presented a novel rough sets approach to estimating application run times. The approach is based on frequencies of attributes appeared in discernibility matrix. The theoretical foundation of rough sets provides an intuitive solution to the problem of application run time estimation on Knowledge Grid. Our hypothesis that rough sets are suitable for estimating application run time in Grid environment is validated by the experimental results, which demonstrate the good prediction accuracy of our approach. The estimation technique presented in this paper is generic and can be applied to other optimization problems.

References

1. D. Talia and M. Cannataro, Knowledge grid: An architecture for distributed knowledge discovery, Communications of the ACM, 2002.
2. I. Foster and C. Kasselman, The Grid: blueprint for a future infrastructure. Morgan Kaufman, 1999.
3. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, The Data Grid: towards an architecture for the distributed management and analysis of large scientific datasets. J. of Network and Comp. Appl., (23):187–200, 2001.
4. A.B. Downey, Predicting Queue Times on Space-Sharing Parallel Computers, In Proceedings of the 11th International Parallel Processing Symposium, pp.209-218, 1997.
5. R. Gibbons, A Historical Application Profiler for Use by Parallel Schedulers. Lecture Notes on Computer Science, Vol. 1297, pp.58-75, 1997.
6. W. Smith , I. Foster, and V. Taylor , Predicting Application Runtimes Using Historical Information, Job Scheduling Strategies for Parallel Processing: IPPS/SPDP'98 Workshop, LNCS 1459, Springer-Verlag, pp.122-142,1998.
7. W. Smith , V. Taylor, and I. Foster, Using Runtime Predictions to Estimate Queue Wait Times and Improve Scheduler Performance, Job Scheduling Strategies for Parallel Processing , LNCS 1659, D.G. Feitelson and L.Rudolph, eds., Springer-Verlag, pp. 202-229,1999.
8. S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, Scheduling high performance data mining tasks on a data grid environment. In Proceedings of Europar, 2002.
9. X. Hu, Knowledge discovery in databases: An attribute-oriented rough set approach, Ph.D thesis, Regina university, 1995.
10. J. Starzyk, D.E.Nelson, K.Sturtz, Reduct generation in information systems, Bulletin of international rough set society, volume 3, 19-22, 1998.
11. J. Komorowski , et al., Rough Sets: A Tutorial, Rough-Fuzzy Hybridization: A New Trend in Decision Making , S.K. Pal and A. Skowron, eds., Springer-Verlag, pp. 3-98, 1998.

A Behavior Characteristics-Based Reputation Evaluation Method for Grid Entities [1]

Xiangli Qu, Xuejun Yang, Yuhua Tang, and Haifang Zhou

School of Computer Science, National University of Defense Technology,
Changsha, China, 410073
cathysmile@eyou.com

Abstract. Reputation provides an operable metric for trust establishment between unknown entities in Grid. Yet, most reputation evaluation methods rarely analyze an entity's past behaviors, which is a deviation from the definition of reputation: an expectation for future behaviors based on past behavior information. Therefore, we propose this behavior-based method for reputation evaluation. The main idea is that: according to reputation evidences from third parties, behavior characteristics such as behavior coherence, behavior inertia etc will be well abstracted, and reputation evaluation will be better guided. Experimental results show that: this method can effectively characterize an entity's behavior, and the final reputation result is reasonable and reliable.

Keywords: reputation, Grid, approximation, behavior coherence factor, behavior inertia degree.

1 Introduction

It shows from survey that, in Grid, collaborations often cross organizational lines, underscoring the significance of inter-organizational trust. The responses showed a high level of communication and collaboration with people from outside the respondent's department (90%), outside their company or organization (82%), and outside their country (69%) [11]. Therefore, in open Grid circumstances with large-scale resource sharing, it is frequent for unknown entities to interact and collaborate with each other. To guarantee smooth ongoing of such cooperation, certain trust relationship must be established among cooperative entities. For completely unknown entities, traditional identity-based trust system obviously cannot work well. In such case, third parties' participation becomes a necessity for trust establishment; meanwhile, Grid's inherent sharing and cooperative characteristics provide an

[1] This work is partially supported by the National 863 High Technology Plan of China under the grant No. 2002AA1Z201, China 863 OS New Technologies Research under the grant No. 2003AA1Z2063 and the Grid Project sponsored by China ministry of education under the grant No. CG2003-GA00103.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 567 – 577, 2005.

excellent support for this. Using shared information in Grid, analyzing evidence of an entity's past behavior, evaluating its trustworthiness, based on which dynamically establishing corresponding trust relationship, this is the basic idea of reputation mechanism in Grid.

For the whole mechanism to succeed, it is crucial to perform a scientific analysis of behavior evidence from third parties and give a reasonable evaluation of an entity's reputation. Yet, in current reputation evaluation, it is usually the evaluation result from third parties that is directly used in combination, with no analysis of original evidence, and as a result analysis of an entity's past behavior is neglected [3] [4] [5] [9]. We believe, for lack of an overall study of an entity's behavior characteristics, it is hard for such methods to give a reasonable and convincing evaluation result. In fact, reputation in itself means the expectation of an entity's future behavior based on information about its past behavior, therefore, analysis of past behavior should be the first step for reputation evaluation. So, we propose this behavior characteristics-based reputation evaluation method. According to the idea of approximation in Fuzzy Set theory [1] [2], we introduce the notion of behavior coherence factor *CF*, as well as behavior inertia degree (including *PID* (Positive Inertia Degree) and *NID* (Negative Inertia Degree)) and behavior deviation *BD*, based on which behavior eigenvector BE is constituted to be an abstract of entity's behavior characteristics so as to guide the evaluation of Grid entities' reputation. Experiments show that, reputation evaluated with this method is reliable and reasonable, which can scientifically represent the overall trend of entities' behavior.

The rest of this paper is organized as follows: a brief introduction of current reputation evaluation method is outlined in section 2; some basic definitions and notations used in this paper are stated in section 3; reputation evaluation methods based on evidence from a single entity and multiple entities are detailed in section 4 and section 5 respectively; specific tests and results are presented in section 6; and finally in section 7 the whole paper is concluded and future work is considered.

2 Related Work

The idea of reputation is firstly used in Electronic Commerce, online communities [15], such as eBay, Amazon and so on. Recently, it is introduced to multi-agent systems, P2P systems and Grid systems [3] [4] [9] [12] [13] [16].

In [12], an EigenTrust mechanism is proposed, which aggregates local trust values according to the notion of transitive trust. During its evaluation process, a kind of normalization is presumed: trust values held by entities are normalized, and their sum is 1. Though this has very good mathematical characteristics: the trust matrix produced is a Markov matrix, and the global trust values just correspond to its left principal eigenvector. But, for the introduction of normalization process, much useful original information is lost.

In [9], Grid Eigen trust, a framework for reputation computing in Grid is introduced. It adopts a hierarchical model, reputation computing is performed from 3 levels: VO, Institution and Entity, that is, an entity's reputation is computed as weighted average of new and old reputation values, an institution's reputation is computed as the eigenvalue of composing entities' reputation matrix, and a VO's reputation is computed as weighted average of all composing institution's reputation value.

In [14] reputation evaluation is based on "Web of Trust". There are two interpretations: Path Algebra Interpretation and Probabilistic Interpretation. Path Algebra Interpretation is performed as follows: 1. Enumerate all (possibly exponential number of) paths between the user and every user with a personal belief in the statement; 2. calculate the belief associated with each path by applying a concatenation function (such as multiplication and minimum value etc.) to the trusts along the path and also the personal belief held by the final node; 3. Combine those beliefs with an aggregation function. Probabilistic Interpretation makes a weighted average of local merged belief and neighbor merged belief.

In [16], Dempster-Shafer theory is adopted as the underlying computing framework, and reputation is combined by means of orthogonal sum of mass function.

Generally speaking, all these existing methods are directly performed on the evaluation result from third parties, with little utilization of those reputation evidences from real interaction. In fact, it is such original information that records an entity's behavior track. If we can perform some analysis of them, it will be easy to characterize an entity's behavior. Moreover, reputation by itself is the expectation of future behavior according to information about past behavior. Therefore, we cannot get a reliable, reasonable and convincing evaluation result unless it is done according to an entity's behavior characteristics. Actually, in Grid it is not difficult to get such reputation related behavior evidence. For example: the Service Negotiation and Acquisition Protocol (*SNAP*) proposed in [8] gives 3 kinds of service level agreement: *RSLA* (Resource Service Level Agreement), *TSLA* (Task Service Level Agreement) and *BSLA* (Binding Service Level Agreement). The 3 agreements supplement each other, clarify a service's provider, user and binding, specify an entity's capability, and concretize interaction context. Therefore, once the reputation value assessed for each cooperator in an interaction is inserted to the above agreement, it can become behavior evidence for us. Storing these evidences to our Grid information system, we can get related evidence for reputation evaluation as needed anytime and anywhere. Our work is done based on this assumption.

3 Related Definitions and Notations

Some definitions and notations used in this paper are given in the following:

Definition 1. *Reputation* In this paper, we define reputation as an expectation of future behavior based on information about past behavior, whose value is within the scope of [0, 1].

Definition 2. *Reputation Evidence* It is a specific assessment of an entity's behavior in some interaction given by its cooperative counterpart according to their agreement, whose value is within the scope of [0, 1], denoted as $R_{<t,e_i,e_j>}$, that is: for the interaction occurring at time (the period of) t between entity e_i and e_j, e_i assesses e_j's behavior reputation as R. The higher the value of R, the trustworthier e_j is. For the set of reputation evidence e_i given to e_j, following the notations in Fuzzy Set,

we use $\{R_0/t_0, R_1/t_1, ..., R_n/t_n\}_{<e_i,e_j>}$ to denote that for the interaction occurring at time (the period of) t_k, e_i assesses e_j's behavior reputation as R_k (elements in the set are ordered according to time).

Definition 3. *Behavior Coherence Factor* denoted as *CF* (*CF*∈ [0,1]), characterizing when an entity cooperates with different entities at different period, to what degree its behavior is coherent. The higher the value of *CF*, the more coherent its behavior is. It includes Time Coherence Factor *TCF* and Entity Coherence Factor *ECF*. *TCF* reflects to what degree an entity's behavior is coherent when interacting with the same entity at different period, while *ECF* reflects to what degree an entity's behavior is coherent when interacting with different entities at the same period.

Definition 4. *Behavior Inertia* reflects the overall changing trend of an entity, including Positive Inertia Degree *PID* and Negative Inertia Degree *NID*. *PID* reflects how the trend that an entity's behavior is more and more satisfying with the increase of time is, whose value is within the scope of [0, 1]. The higher the value of *PID*, the stronger an entity's better-behaving trend is, while *NID* denotes an entity's worse-behaving trend.

Definition 5. *Behavior Deviation* denoted as *BD*, reflects to what degree an entity's behavior fluctuates. BD∈ [0, 1].

Definition 6. *Trustworthy entity* We believe a trustworthy entity should embody the following characteristics: the satisfying degrees given by cooperative counterparts are generally high; its behavior is basically stable: behaves basically coherently when interacting with different entities and at different time; the overall behavior embodies a well-evolving trend.

4 Reputation Evaluation Based on Evidence Set from a Single Entity

To begin with, we will evaluate entity e_j's reputation based on evidence set $E = \{R_0/t_0, R_1/t_1, ..., R_n/t_n\}_{<e_i,e_j>}$ from a single entity e_i.

4.1 Behavior Characterization

According to the fact that reputation decays with time, the most recent reputation evidence R_n/t_n has the most deciding effect in the whole evidence set. Therefore, we choose it as a criterion to construct a standard set $E_0 = \{R_n/t_0, R_n/t_1, ..., R_n/t_n\}_{<e_i,e_j>}$. For the inherent fuzzy character of reputation, the two set can both be viewed as a fuzzy set.

1. *TCF*

For evidence set from a single entity, we will study the behavior coherence for interactions with the same entity during the period selected evidence set spans, that is,

TCF is used to evaluate behavior coherence. Since each evidence in the set is given from a same evaluator, therefore the criterion to evaluate each interaction is relatively fixed; under this precondition it is reasonable to compare reputation values of each time. In the specific coherence analysis, we borrow the idea of approximation from Fuzzy Set theory. Since in Fuzzy Set theory, approximation characterizes how two fuzzy sets approximate each other [1], to some degree the approximation of the two sets E and E_0 reflects the time coherence of an entity's behavior. In the specific calculation, we can choose Hamming approximation, Euclid approximation and so on:

For Hamming approximation, we will get:

$$TCF = N(E,E_0) \overset{\Delta}{=} 1 - \frac{1}{n+1}\sum_{i=0}^{n}\left|E(R_i) - E_0(R_i)\right| = 1 - \frac{1}{n+1}\sum_{i=0}^{n}\left|R_i - R_n\right| \quad (1)$$

For Euclid approximation, we will get:

$$TCF = N(E,E_0) \overset{\Delta}{=} 1 - \frac{1}{\sqrt{n+1}}\left(\sum_{i=0}^{n}(E(R_i) - E_0(R_i))^2\right)^{1/2} = 1 - \frac{1}{\sqrt{n+1}}\left(\sum_{i=0}^{n}(R_i - R_n)^2\right)^{1/2} \quad (2)$$

2. *Inertial Degree*

PID is used to describe an entity's well-behaving trend for the spanned time period. Its calculation is very simple: just compare neighboring reputation values one by one, if the count of instances where the successor has a greater value than its predecessor is m, then:

$$PID = m/n \quad (3)$$

As opposed to the above, if the count of instances where the successor has a smaller value than its predecessor is *l*, then $NID = l/n$.

If *PID* is greater than *NID*, it shows that the entity's reputation embodies a steadily increasing trend, and the more closely *PID* approaches 1, the more evident this trend is; If *PID=NID*, it shows that the entity's reputation fluctuates upward and downward, and the overall trend is not clear; if *PID* is smaller than *NID*, it shows that the entity's reputation embodies a steadily decreasing trend, and the more *PID* approaches 0, the more evident this trend is. Noted that, the sum of *PID* and *NID* is not always 1, since sometimes the neighboring reputation value is equal.

3. *BD*

BD reflects, in standard of R_n / t_n , other evidences' average deviation, which is calculated as Equation (4) shows:

$$BD = \frac{\left|\sum_{i=0}^{n-1} R_n - R_i\right|}{n} \quad (4)$$

4. Behavior Eigenvector *BE*

For single entity based evidence set, *BE=(TCF, PID, NID, BD)*.

4.2 Reputation Evaluation

By means of *BE*, we characterize entity e_j's behavior when interacting with entity e_i, based on which we will evaluate e_j's reputation as Equation (5) shows:

$$R_{<e_i,e_j>} = \begin{cases} TCF * R_{wavg} + min(BD,\ 1 - R_{wavg}) & PID > NID \\ TCF * R_{wavg} & PID = NID \\ max(TCF * R_{wavg} - BD,\ 0) & PID < NID \end{cases} \quad (5)$$

Herein, R_{wavg} denotes the time-decaying weighted average of all reputation evidences from set E, and its calculation is:

$$R_{wavg} = \frac{\sum_{i=0}^{n} D(t_i) * R_i}{\sum_{i=0}^{n} D(t_i)} \quad (6)$$

Wherein $D(t)$ is a time decaying function. Here, we use the fact that reputation will decay with time.

Obviously, $0<TCF\leq 1$, and $min(BD,\ 1-R_{wavg}) \leq 1\text{-}R_{wavg}$, so $0 < TCF * R_{wavg} + min(BD,\ 1-R_{wavg}) \leq R_{wavg} + 1 - R_{wavg} = 1$; Similarly, it is easy to get that $0 < TCF * R_{wavg} < 1$ and $0 \leq max(TCF * R_{wavg} - BD, 0) < 1$; therefore $R_{<e_i,e_j>} \in [0,1]$ and $R_{<e_i,e_j>}$ is a kind of reputation. From Equation (5) and (6), it can be seen that, this method considers such behavior characteristics as coherence, inertia and deviation, as well as the time decaying characteristics of reputation.

5 Reputation Evaluation Based on Evidence Sets from Multiple Entities

Reputation evaluation based on evidence sets from multiple entities is established on the basis of evaluation based on evidence set from a single entity. Before evaluating, we should first perform evaluation on each evidence set from a single entity according to the method given in section 4. Note that, to guarantee the comparability of evidence sets, and with the consideration of the fact that reputation decays with time, we should select evidence sets with approximately same time span when acquiring evidence sets from Grid information service. Suppose that, we get *p* evidence sets from different entities $e_0, e_1 \cdots e_{p-1}$, which constitute the reputation evidence set

for entity e, where evidence set from entity e_k is denoted as $E_k = \{R_0/t_0, R_1/t_1, ..., R_n/t_n\}_{<e_k,e>}$. Noted that, here t_i is a period of time. In fact, some process should be performed on the original evidence set to get such evidence set with time correspondence: if there are several evidences in the original evidence set during the period of t_i, R_i is computed as their time-decaying weighted average, as is illustrated in Equation (6); if there is no evidence during the period of t_i, R_i will take the value of the final reputation result for corresponding single evidence set, which is computed according to Equation (5).

5.1 Behavior Characterization

Behavior characterization is still the first step in our reputation evaluation. For evidence sets from multiple entities, our analysis mainly focuses on entity coherence. Similar to the evaluating method for time coherence, it is necessary to find a standard set for comparison. This is chosen as follows: according to past interaction experience, choose a most trustworthy entity, suppose it is e_0, then we will use its evidence set as the standard set. If it happens that all the p entities are unknown, randomly select one as the standard. The next step is analyzing the entity coherence factor for each evidence set as compared to E_0, taking E_m as an example:

Since there is some difference between assessments of the same behavior from different entities, we adopt a relaxed model in ECF calculation though the basic idea still comes from approximation in Fuzzy Set theory. We make comparison of coarser granularity, the specific computing step is as follows:

Step 1. Compute the difference set ΔE:

$$\Delta E_{<m,0>} = \left\{ \Delta R_i \Big/ t_i \right\}_{i=0,1...n} = \left\{ \left| R_{i<m>} - R_{i<0>} \right| \Big/ t_i \right\}_{i=0,1...n} \tag{7}$$

Step 2. Perform a classification of ΔE according to the criterion $< \sigma_{\text{coherent}}, \sigma_{\text{fuzzy}}, \sigma_{\text{discoherent}} >$ (where $0 < \sigma_{\text{coherent}} < \sigma_{\text{fuzzy}} < \sigma_{\text{discoherent}} < 1$), and divide ΔE into 4 subsets: coherent subset E_c, comparatively coherent subset E_{half-c}, comparatively discoherent subset $E_{half-disc}$ and discoherent subset E_{disc}:

$$\Delta R_i \in \begin{cases} E_c & \Delta R_i \in [0, \ \sigma_{cohrent}) \\ E_{hafl-c} & \Delta \mathrm{R}_i \in [\sigma_{cohrent}, \ \sigma_{fuzzy}) \\ E_{half-disc} & \Delta R_i \in [\sigma_{fuzzy}, \ \sigma_{discohrent}) \\ E_{disc} & \Delta R_i \in [\sigma_{discohrent}, \ 1] \end{cases} \tag{8}$$

Step 3. Compute ECF: according to the weighted coherence vector $< \omega_{\mathrm{c}}, \omega_{\mathrm{half\text{-}c}}, \omega_{\mathrm{half\text{-}disc}}, \omega_{\mathrm{disc}} >$ $(1 \geq \omega_{\mathrm{c}} > \omega_{\mathrm{half\text{-}c}} > \omega_{\mathrm{half\text{-}disc}} > \omega_{\mathrm{disc}} \geq 0)$, *ECF* between E_0 and E_m is computed as follows:

$$ECF_{<m,0>} = \frac{\omega_c * \|E_c\| + \omega_{half-c} * \|E_{half-c}\| + \omega_{half-disc} * \|E_{half-disc}\| + \omega_{disc} * \|E_{disc}\|}{\|\Delta E_{<m,0>}\|} \tag{9}$$

5.2 Reputation Evaluation

With *ECF*, the method we use to evaluate entity *e*'s reputation is given in Equation (10) (where $ECF_{<0,0>} = 1$):

$$R_{<e_0,e_1,\ldots,e_{p-1},e>} = \frac{\sum_{i=0}^{p-1} R_{<e_i,e>} * ECF_{<i,0>}}{\sum_{i=0}^{p-1} ECF_{<i,0>}} \tag{10}$$

6 Experimental Results

For the behavior characteristics-based reputation evaluation method stated above, we performed a series of simulated experiments. Results show that our method makes a good characterization of entity's past behavior, and the reputation result is relatively reliable and reasonable. In the following experiments, each evidence set from a single entity includes 10 evidences, and the 10 evidences are distributed within same time interval. In the experiment, we use Hamming approximation method, as is shown in Equation (1), to calculate *TCF*. And the time decaying function *D(t)* adopted is:

$$D(t_i) = e^{-\sqrt{n-i}}, i = 0,1,\ldots 9 \qquad n = 9 \tag{11}$$

6.1 Experiments for Evidence Set from a Single Entity

Since the evaluation of reputation is established on the basis of information about past behavior, in our experiments various kind of behavior modes (mainly focusing on behavior coherence and evolving trend) are used, so as to test the effectiveness and reasonability of this behavior characteristics-based reputation evaluation method. In the experiments for evidence set from a single entity, we simulate 7 kinds of behavior mode. In Table 1, results from our method and from commonly used time-decaying weighted average method are compared:

From Table 1, it can be seen that our method has taken each entity's behavior characteristics into account: those entities with high coherence and obvious well-behaving trend get higher reputation. Compared with time-decaying weighted average method, our method can characterize an entity's behaving trend as a whole,

scientifically integrate information about past behavior, and have a reliable prediction of future trend. It is not difficult to see that, this method well matches what we defined as a trustworthy entity in section 3: only those entities with coherent behavior and well-evolving trend are trustworthy entities.

Table 1. Experimental results for evidence set from a single entity

Entity	Behavior Mode	Original evidence										Evaluation result	
		t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	Our method	Weighted average method
0	Basically Stable	0.81	0.82	0.81	0.81	0.82	0.82	0.83	0.83	0.82	0.82	*0.818*	0.821
1	Completely Stable	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	*0.8*	0.8
2	Steadily Increasing	0.7	0.7	0.75	0.75	0.8	0.8	0.8	0.85	0.85	0.85	*0.845*	0.827
3	Continuously Increasing	0.78	0.8	0.83	0.85	0.87	0.88	0.89	0.90	0.91	0.92	*0.910*	0.898
4	Fluctuating	0.8	0.7	0.8	0.7	0.7	0.8	0.8	0.7	0.7	0.7	*0.724*	0.762
5	Steadily Decreasing	0.84	0.83	0.83	0.82	0.82	0.82	0.81	0.81	0.8	0.8	*0.772*	0.807
6	Continuously Decreasing	0.92	0.91	0.9	0.89	0.88	0.87	0.85	0.84	0.83	0.82	*0.741*	0.841

6.2 Experiments for Evidence Sets from Multiple Entities

In experiments for evidence sets from multiple entities, the selected classification criterion $<\sigma_{\text{coherent}}, \sigma_{\text{fuzzy}}, \sigma_{\text{discoherent}}>$ is <0.1,0.2,0.3>, and the weighted coherence vector $<\omega_{\text{c}}, \omega_{\text{half-c}}, \omega_{\text{half-disc}}, \omega_{\text{disc}}>$ is <1,0.6,0.3,0>. Next, we will take the 7 evidence sets from Table 1 as an example to perform our evaluation. Deem that the entity of "basically stable" behavior mode is the most trustworthy entity, taking its evidence set as a standard, we get the following *ECF* results:

$$ECF_{<1\,0>}=1;\ ECF_{<2\,0>}=0.92;\ ECF_{<3\,0>}=0.96;\ ECF_{<4\,0>}=0.8;\ ECF_{<5\,0>}=1;\ ECF_{<6\,0>}=0.96$$

Refer to the reputation distribution curve in Figure 1, we can see that the ECF result above is reasonable.

And according to Equation (10), the final result of reputation is: 0.808.

7 Conclusions and Future Work

In this paper, we give a behavior characteristics-based reputation evaluation method for Grid entities. Its underlying idea is a good match to the definition of reputation: an expectation of future behavior according to information about past behavior. This

method focuses on analysis of an entity's behavior coherence (including time coherence and entity coherence) and inertia, based on which reputation evaluation for evidence set from a single entity and multiple entities is performed. Experiments show that, this method can well characterize an entity's behavior and get reliable and reasonable evaluation results.

Currently, the analysis of behavior characteristics for multiple entities-based evidence sets only involves behavior coherence between entities, therefore we hope that more behavior characteristics can be abstracted as a better guide for reputation evaluation for evidence sets from multiple entities.

And as mentioned in Section 2, our work is largely based on the assumption that we have real reputation data of each participant in an interaction. For all the processing relating to this kind of data, such as gathering, storing, accessing etc., we propose to implement a Grid Reputation Service as proposed in [17]. An implementation of this service is another work to be done.

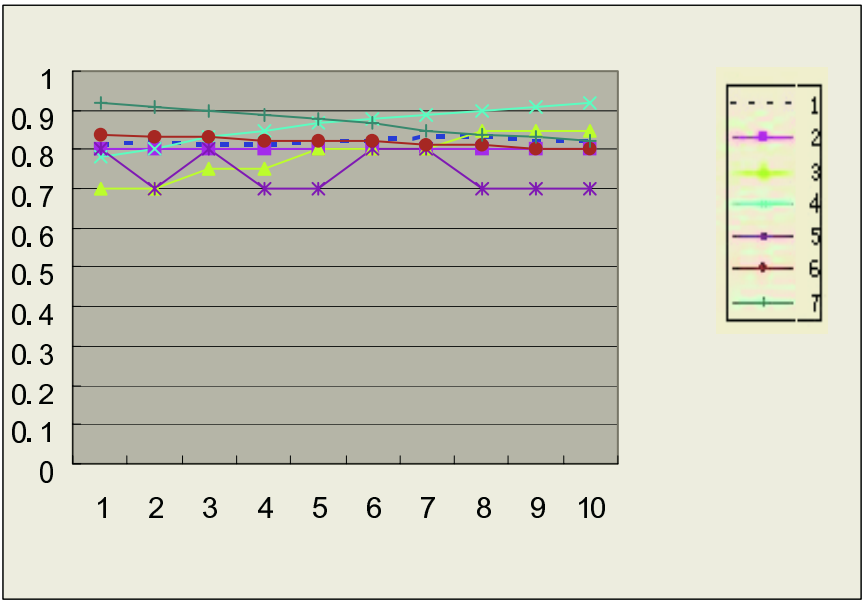


Fig. 1. The data distribution of 7 types of evidences (From 1 to 7 are: basically stable, completely stable, steadily increasing, continuously increasing, fluctuating, steadily decreasing and continuously decreasing.)

References

1. L.B. Yang and Y.Y.: Gao The Theory and Application of Fuzzy Sets (Second Edition). South China University of Technology Press (1996)
2. C.Z. Luo: An Introduction to Fuzzy Sets (I). Beijing Normal University Press (1989)

3. Beulah Kurian Alunkal, Ivana Veljkovic, Gregor von Laszewski, and Kaizar Amin1: Reputation-Based Grid Resource Selection. AGridM 2003: Workshop on Adaptive Grid Middleware (September 28, 2003)
4. Farag Azzedin and Muthucumaru Maheswaran: Evolving and Managing Trust in Grid Computing Systems. Proceedings of the 2002 IEEE Canadian Conference on Electrical Computer Engineering
5. G. Zacharia and P. Maes: Trust Management through Reputation Mechanisms. Applied Artificial Intelligence 14 (2000) 881~907
6. S. P. Marsh: Formalising trust as a computational concept. Ph.D. Thesis (1994)
7. Michael Schillo, Petra Funk and Michael Rovatsos: Using Trust for Detecting Deceitful Agents in Artificial Societies. Applied Artificial Intelligence, 14 (2000) 825~848
8. K.Czajkowski, I.Foster, C.Kesselman: "SNAP:A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems", Lecture Notes in Computer Science, Vol. 2537. Springer-Verlag (2002) 153~183
9. Beulah Kurian Alunkal: Grid EigenTrust: A Framework for Computing Reputation in Grids. MS thesis, Department of Computer Science, Illinois Institute of Technology (Nov, 2003)
10. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis: The Role of Trust Management in Distributed Systems Security. In Vitek, J., Jensen, C.D., eds. Chapter in Secure Internet Programming: Security Issues for Mobile and Distributed Objects. Lecture Notes in Computer Science, Vol. 1603. Springer-Verlag (1999)
11. Markus Lorch, Dennis Kafura: Grid Community Characteristics and their Relation to Grid Security. VT CS Technical Report, 2003. http://zuni.cs.vt.edu/publications/draft-ggf-lorch-grid-security-version0.pdf
12. Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina: The EigenTrust Algorithm for Reputation Management in P2P Networks. In Twelfth International World Wide Web Conference, 2003, Budapest, Hungary, May 20~24 2003. ACM Press. http://www.stanford.edu/_sdkamvar/papers/eigentrust.pdf.
13. E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante: Reputation-based Method for Choosing Reliable Resources in Peer-to-peer Networks. In Proc. of the 9th ACM Conference on Computer and Communications Security (2002)
14. M. Richardson, R. Agrawal, and P. Domingos. Trust management for the Semantic web. In Proceedings of the Second International Semantic Web Conference (2003) 351~368
15. B. Yu and M. P. Singh. A Social Mechanism of Reputation Management in electronic Communities. In Cooperative Information Agents (2000) 154~165
16. B. Yu and M. P. Singh. An Evidential Model of Distributed Reputation Management. In Proceedings of the first international joint conference on Autonomous Agents and Multiagent Systems: Part 1 (2002) 294~301
17. Xiangli Qu, Nong Xiao, Guang Xiang, Xuejun Yang: Reputation-aware Contract-supervised Grid Computing. GCC Workshops 2004. Lecture Notes in Computer Science, Vol. 3252. Springer-Verlag-Berlin Heidelberg (2004) 44~51

Dynamic Policy Management Framework for Partial Policy Information

Chiu-Man Yu and Kam-Wing Ng

Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Shatin,
New Territories, Hong Kong SAR
{cmyu,kwng}@cse.cuhk.edu.hk

Abstract. A Grid can support organized resource sharing between multiple heterogeneous Virtual Organizations. A Virtual Organization can be considered to be an administrative domain consisting of hosts from different networks. The dynamic VO membership organization of Grid computing and heterogeneous VOs are major challenges to policy management. We propose a Dynamic Policy Management Framework (DPMF) to resolve the problems. DPMF groups VOs of same policy framework to form a virtual cluster. Policy management is divided into inter-cluster heterogeneous policy management, and intra-cluster homogeneous policy management. Inside a virtual cluster, policy agents of the VOs form trust relationship hierarchy for distributing the works of policy conflict analysis. Conflict Analysis with Partial Information (CAwPI) mechanism is developed to provide an approach to analyze policy conflict in open environments without complete policy information.

1 Introduction

Grid computing enables computers on different networks to share their resources in an organized way. Authorized users can deploy the resources as if they were in the same organization. This resource sharing environment is called a Virtual Organization (VO) [3]. A Virtual Organization applies coherent resource sharing mechanism and authorization policy management framework to enable organized resource sharing in the VO. It would be desirable for a Grid to be able to manage multiple VOs. Regarding to the dynamic nature of the Grid, Grid membership is dynamic. VOs can join or leave the Grid environment throughout the Grid lifetime. Besides, regarding to the heterogeneous nature of the Grid, the VOs may use different policy frameworks. The dynamic and heterogeneous natures impose challenges to policy management on Grids. Traditional security policy frameworks [5][7][8] deal with security policy management inside a VO. There is still little research on policy management for multiple VOs. Due to the increasing popularity of Grid computing, it is likely that there will be a large number of application systems based on Grid computing environments in the future. Each Grid application system forms a VO. With the use of different grid development toolkits, the VOs may deploy different security policy frameworks. To enable secure interaction of heterogeneous VOs, frameworks and mechanisms are

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 578 – 588, 2005.

needed to manage security policies across the VOs. We have proposed a Security Policy Dynamic Management Framework (DPMF) [9] to handle this problem. In this paper, we present the DPMF's architecture and its mechanism of policy conflict analysis with partial information.

A Grid may consist of multiple VOs. Each VO is considered to be an individual administrative domain. Each administrative domain can decide on the resources and information for sharing. The domains set their own authorization policies to limit access to their resources. The access limitation can include domains, users, time, and other tailor-made conditions.

In a Grid computing environment, a user may plan to perform tasks which involve access to resources on multiple domains. To determine the limitations to perform the tasks, the user needs to check the authorization policies of the resource owners. Therefore the user needs to retrieve all the policies related to the user and the resources. The user then performs conflict analysis on the policies to find out the limitations.

In a Grid computing environment, some VO domains may refuse to share their policy information to other (or certain) domains. If some resource owners do not disclose their authorization policies to the user, the user can only get partial policy information. The user would have to perform conflict analysis with partial policy information.

In this paper, we present our approach of conflict analysis with partial information. The approach includes construction of substitutions to uncertain policies, and conflict detection with partial policy information.

This paper is structured as follows. Section 1 introduces the target problems and related works. Section 2 reviews our proposed Dynamic Management Framework for policy management in open Grids. Section 3 presents the "conflict analysis with partial information" mechanism. Section 4 makes the conclusions.

2 Dynamic Policy Management Framework (DPMF)

Dynamic Policy Management Framework (DPMF) is a hierarchical framework which aims to support "dynamic policy management" and "heterogeneous policy conflict analysis" for Grid environments of multiple VOs. It contains a number of "Policy Agents" (PA), "Policy Processing Units" (PPU) and a "Principal Policy Processing Unit" (P-PPU). DPMF deploys PAs to divide VOs into virtual clusters according to their security policy frameworks. Conflict analysis can be performed homogeneously in a virtual cluster, or can be performed heterogeneously through the P-PPU. The PAs' virtual connection architecture inside a virtual cluster is constructed hierarchically according to trust relationship so that policy management tasks can be distributed to PAs.

In the Grid environment model for DPMF, there is one Grid Operator and a number of VOs. The Grid Operator coordinates meta-VO interaction or interoperation among the VOs. The Grid Operator also maintains meta-VO level policies. Each VO consists of a number of nodes which can be service providers, or service requesters. Each VO has a policy server. The VOs' policy servers and the Grid Operator are PDPs (Policy Decision Points). The service providers and service requesters on the VOs are PEPs (Policy Enforcement Points). A PDP is the point where the policy decisions are made; and a PEP is the point where the policy decisions are actually enforced [6].

2.1 Framework Architecture

Figure 1 illustrates the DPMF's hierarchical architecture. In DPMF, each VO needs to own a PA. The Grid Operator owns a P-PPU. PAs of the same security policy framework would form a virtual cluster. One of the PAs in each virtual cluster would be elected as PPU. A PPU performs runtime conflict analysis for homogeneous security policy framework, whereas the P-PPU is used to perform heterogeneous security policy conflict analysis across different security policy frameworks. In DPMF, the service providers are PEPs; whereas the PAs, PPUs, and P-PPUs are PDPs.

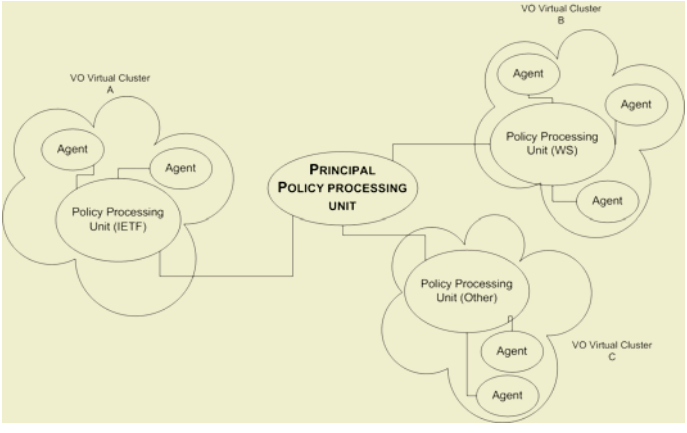


Fig. 1. DPMF architecture

All PPUs are connected to the single P-PPU. PAs are connected to either the PPU or other PAs. PAs can access the information of their local security policies. A PPU can access the information of its local security policies, global security policies, and knowledge of local conflict analysis. A P-PPU can access the information of global policies, and the knowledge of heterogeneous policy conflict analysis. The PA needs to be able to access the VO's security policies. If the VO already has a PDP (or policy server), the PA is likely to be run on the PDP. Otherwise, the PA can be run on a node which has privilege to access the security policy repositories in the VO.

PAs of the same "security policy framework model" would form a virtual cluster. For example, Web Services policy framework (WSPolicy) [1] and IETF security policy framework [4] are different framework models. A newly initialized PA can ask the P-PPU for which virtual cluster to join. PAs in a virtual cluster elect one of them to be the PPU. PPU is also a kind of PA.

In an ideal trust relationship, if all PAs in the same virtual cluster trust one of the PAs, then the PA can become the PPU of the cluster. However, usually the ideal trust relationship cannot be achieved. Some VOs do not trust other VOs, or none of the VOs are trusted by all other VOs in the virtual cluster. Therefore, DPMF needs a PPU election mechanism in the non-ideal trust relationship. The election mechanism selects a PA with the most number of supporters to be PPU. The PPU connects to PAs

according to the trust relationships of the PAs. The PPU may be untrusted by some PAs such that the PPU is unable to retrieve the security policy information from some PAs. Therefore the PPU may need to perform conflict analysis with partial information. Simultaneously, the PPU maintains the PA trust relationship hierarchy in the virtual cluster. It finds out the most suitable PA for a conflict analysis task according to the hierarchy, and it can delegate the task to the PA.

2.2 Policy Agent (PA) Trust Relationship Hierarchy

In a virtual cluster, not all PAs are virtually connected to the PPU directly by default. The PA virtual connection hierarchy depends on their trust relationships.

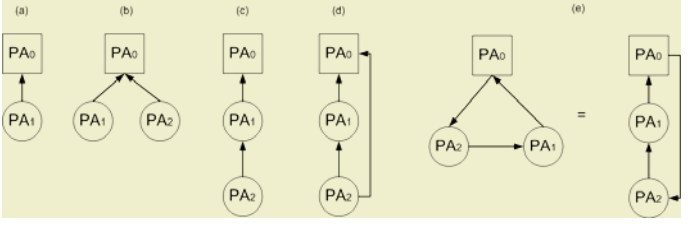


Fig. 2. Examples of PA trust relationships

Figure 2 shows several examples of trust relationships between PAs. The symbol "←" represents the direction of trust. The PA in the square box is the subject PA. We transform the PA trust relationship diagrams into two-way "be trusted" and "to trust" expressions. The trust relationships for Figure 2 can be expressed in the followings:

(a) $PA_0 \leftarrow \{PA_1\}$
(b) $PA_0 \leftarrow \{PA_1, PA_2\}$
(c) $PA_0 \leftarrow \{PA_1\}$
(d) $PA_0 \leftarrow \{PA_1, PA_2\}$
(e) $PA_0 \leftarrow \{PA_1\}$ and $PA_0 \rightarrow \{PA_2\}$

PA trust relationship hierarchy is to record trust relationships between PAs for distributing policy management tasks. DPMF deploys a trust model in which a trust relationship is non-symmetric and non-transitive. Therefore, the PA trust relationship hierarchy is reduced to sets of "be trusted" (symbol "←") PAs and sets of "to trust" (symbol "→") PAs. In the examples in Figure 2, the trust relationships of the subject PA in the diagrams can be record as the trust relationships as shown above. Using the information of PA trust relationship hierarchy, the PPU can find the subject PA which is trusted by most PAs in a conflict analysis task. Then the PPU can delegate the task to the subject PA to perform conflict analysis. This enables the workload of conflict analysis to be distributed in a virtual cluster.

In DPMF, each VO owns a PA which can access to the policy information of the VO domain and the VO's service providers. The PA is an access point to the policy repositories in the VO. DPMF defines P-PPU and PPU to manipulate policy management services which include:

1. *Notification of policy update*
   When a service provider or a VO has its policy information updated, its representative PA needs to inform the PPU for the update. The PPU checks if the policy update involves in current service requests. The PPU needs to notify a service requester if the request's request is not valid after the update. The PPU supports the notification service for service requests which involve services in two or more VOs in the same virtual cluster. The P-PPU supports the notification service for service request where the target service is on a VO in a different virtual cluster.
2. *Conflict analysis of policies*
   When a host wants to run a task which involves services on multiple VOs, it sends the request to the PPU. The PPU requests the PAs of target services for authorization policy information. The PPU checks if there is conflict within the set of authorization policies related to the service request. Policy decision is made after the conflict detection. Since PAs in a virtual cluster use the same policy framework, the conflict analysis performed by the PPU is homogeneous in terms of policy model. On the other hand, the P-PPU handles conflict analysis for heterogeneous policies.

Table 1. Classification of agents which support policy management services

Agent	Scope of target service requests
P-PPU	Services on VOs which are in different virtual clusters.
PPU	Services on multiple VOs which are in same virtual cluster.
PA	Subset of PPU's scope while the PA has authorization to the policy information involved in the service request.

DMPF deploys the PA trust relationship hierarchy to delegate policy management services of PPU to other PAs in same virtual cluster. According to the PA trust relationship hierarchy, the PPU can know the trusting PA(s) of each PA, that is, the PA(s) which trust that PA. A trust relationship in DPMF represents authorization to policy information. Therefore the PPU can know the authorization of policy information of each PA. To distribute workload among PAs in a virtual cluster, the PPU delegates a number of service requests and corresponding policy management services to suitable PAs. Table 1 lists the three kinds of agents and their respective scopes of service request. After receiving a service request involving services on multiple VOs, PPU can delegate the policy management task through the following procedure.

1. Search the PA trust relationship hierarchy for the PA which has greatest authorization of policy information related to the request.
2. Request the PA to handle a service request.
3. The PA replies to the PPU for accept or rejection. If the PA rejects, then the PPU handles the service request as normal.
4. If the PA accepts, then the PPU sends the service request to the PA, and notify the service requester for the assignment of a request handler.
5. The PA supports conflict analysis, policy decision, and update notification services for that service request.

3 Policy Conflict Analysis with Partial Information (CAwPI)

When some PAs do not trust the PPU, the PPU is not able to retrieve certain security policy information from the PAs. For example, an individual from a VO wants to perform interoperation with individuals from VOs in the same virtual clusters. So the PAs of the VOs request the PPU to perform conflict analysis. All of the PAs trust the PPU except one. The PPU cannot retrieve the policy information from the untrustful PA. The PPU can only perform policy conflict analysis with policy information from the trustful PAs. Therefore we need to look into methods to perform conflict analysis with partial policy information.

The main idea of our current approach of partial information conflict detection is that the PPU performs conflict analysis with the known security policies, generates substitutions for unknown policies, finds out a set of conditions which can cause conflicts, then sends the set of conditions to the untrustful VO for its checking. When the PPU is firstly initialized, it constructs a policy template database which contains a set of popular policy templates. The templates are generated by reading policies of VOs (which trust the PPU) in the same virtual clusters. When there is a request to perform conflict analysis with partial information. The PPU retrieves all possible policies templates from the policy template database to substitute for the non-retrieved policy information. It recursively performs conflict analysis with substitutions of different (sets of) policy templates. This is the critical process of partial information conflict analysis. In this section, we describe the model of policy templates, and the conflict analysis mechanism using policy templates for partial information.

3.1 Assumptions, System Model, and Policy Model

Two assumptions are made in DPMF related to the trust relationship between VOs and the policy information sharing:

1. "A PA trusts a PPU or PA" means that the PA will disclose its complete authorization policy information to the PPU or PA.
2. A PA can obtain policy sets of service providers in its VO domain.

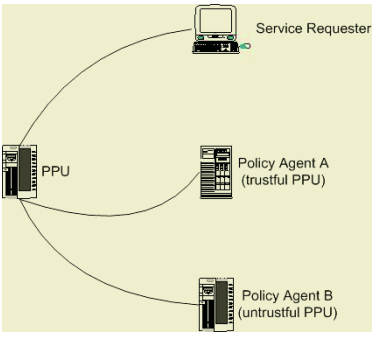


Fig. 3. An example of system model for CAwPI

Figure 3 shows an example of system model for CAwPI. The system model describes the target generic system setting for the conflict analysis mechanism. The model describes the necessary principals for a service request which requires conflict analysis with partial information. In the example, a service requester requests services on two (or multiple) VOs. Policy Agent A and Policy Agent B handle security policies of the two VOs respectively. The service request to multiple VOs would require a conflict analysis between the service requester and the VOs involved. A PPU is responsible to perform the conflict analysis. The PPU needs to retrieve policy information from the Policy Agents. If one or more PA(s) does not trust the PPU and hence does not share its policy information to PPU, the PPU needs to perform conflict analysis with partial information.

DPMF defines a skeletal policy model in order to inter-exchange with other existing policy models. The policy model is mainly a reduced version of IETF Policy Core Information Model [4] with the addition of an extension option. The policy model in DPMF defines that each service provider maintains an authorization policy set. Each domain can also maintain an authorization policy set which applies to all service providers in the domain. A policy set consists of a number of individual policies. Each policy has four components. They are *Condition Set, Action Set, Target Identity, and extension element(s)*.

The *Condition Set* is a collection of conditions with AND or OR relationship. Each condition is a Boolean expression which can result into true (positive) or false (negative) logic. The *Action Set* is a collection of actions with AND relationship. The actions can be defined to be performed in one-by-one order or simultaneously. The Policy Rule in the model is authorization model. Therefore, the Action is to define if a Target Identity is allowed to access a resource or if a Target Identity is prohibited to access a resource. The *Target Identity* is a credential which can be represented by a certificate to recognize a user, or just a login name in the local domain.

The *Extension Element(s)* is used to record elements which cannot be represented by Condition Set or Action Set. This reserves rooms for compatibility with various policy models. However, to activate this option, all principals for a conflict analysis task (Service requester, PAs, PPU) need to be able to recognize the extension element(s).

3.2 Policy Template

The mechanism of CAwPI is to generate a set of policy templates for substitution of unknown policy information. The policy template model includes the skeletal policy model of DPMF, an Evaluation Element Set, and a corresponding Priority Set. The Evaluation Element Set stores evaluation elements which are attributes of the policy owner. The attributes are defined by the PPU. They probably include the type of service providers, the type of Virtual Organizations, security levels, etc. The Priority Set stores the order of importance of the evaluation elements.

Condition =
when (<*time expression variable*>) [*occurred*, *occurring*] (<*event expression variable*>)
Action =
do ([*Permit*, *Prohibit*] to access <*resource variable*>)
Target Identity = <service requester identity>

Extension Element Set
Evaluation Element Set
Priority Set

The format of a policy template is shown above. The Condition Set, Action Set, and Extension Element Set are learnt from policy information provided by trustful PAs. The Evaluation Element Set and Priority Set are defined by the PPU. In a CAwPI task, PPU would deploy certain policy templates to generate substitution policies to substitute unknown policies. When a policy template is used in a conflict analysis task, substitution policies are generated by setting the Target Identity to be the service requester's identity.

3.3 Conflict Detection with Partial Information

The conflict detection of DPMF currently targets on authorization policy conflicts. An authorization policy defines if a Target Identity is permitted or prohibited to access a resource. The policy model in DPMF is described in Section 3.2. A conflict occurs when two policies have same Condition Set but opposite authorization actions. Positive authorization refers to permission to access while negative authorization refers to prohibition to access.

policy1 →← policy2
when (*time expression*) [occurred, occurring] (*event expression*)
→ do ([**Permit/Prohibit]** to access *resource*) for <Target Identity>

The above expression shows a generic conflict model for policy conflict between two policies. The symbol "→←" represents conflict relationship, which is symmetric. The expression shows that policy1 and policy2 have the same condition but resulting in opposite signs of authorization on the *resource* for the Target Identity. This kind of conflict is a modality conflict [2].

The conflict detection mechanism in DPMF is performed in three phases:

1. *Collect policy information.*
2. *Perform conflict detection on the policies.*
3. *Generate conflict results.*

For a conflict detection task involving service providers on VOs which all trusts the PPU, the respective PAs would provide the policies related to the service requester identity. The provided policies include the policy set of the service providers and that of the VO domain. PPU can then transverse the policies to detect modality conflict between any two policies. DPMF currently cannot support conflict analysis if all the involved PAs do not trust the PPU.

After collection of policy information, the PPU can detect modality conflicts in the following steps:

1. Target a *resource* requesting from service requester.
2. Transverse the policies to select the ones whose resource field is *resource*.
3. Select the pairs in which the authorization signs are opposite.
4. Check if their Condition Sets have intersection or not.
5. If yes, then conflict is detected.

If some PAs do not trust the PPU, then the Conflict Analysis with Partial Information (CAwPI) would be initialized at the "*Collect policy information*" phase. The mechanism of CAwPI is different from normal conflict detection mechanism in DPMF, but it is also divided into three phases.

1. *Collect policy information*
   PA(s) which trusts the PPU would send the related policy information to the PPU. For PA(s) which does not trust the PPU, the PPU needs to deploy policy templates to generate substitution policies. The PPU uses the Evaluation Element Set and Priority Set for the selection of policy templates. The policy generation procedure can be primarily divided into two steps:

   (i) Given a predefined priority threshold, for priority from high to low, target the evaluation elements one by one. Select policy templates which have same value of an evaluation element to the untrustful PA.
   (ii) Given a predefined priority threshold and similarity threshold of individual evaluation elements, target the evaluation elements one by one. Select policy templates which have similar values of an evaluation element to the untrustful PA.

2. *Perform conflict detection on the policies*
   The detection procedure of CAwPI is mostly similar to the normal conflict detection procedure. However in CAwPI, when there is a substitution policy involved in a conflict, the policy would be stored in a Conflict Policy Set.

3. *Generate conflict results*
   PPU sends the Conflict Policy Set to the untrustful PA(s). The set contains policies which can cause conflicts to the service request. PPU queries the untrustful PA to see if the PA's VO domain and its respective service provider(s) have any one of the policies. There can be three possible replies from the PA:

   (i) If the PA replies "no", PPU concludes that no conflict is found.
   (ii) If the PA replies "yes", PPU concludes that conflict is found.
   (iii) IF the PA does not reply, PPU concludes that there is insufficient information for the conflict analysis.

Every PA can collect policy information to train its policy template set. PPU should be able to exchange policy template set information between trusting PAs.

4 Conclusions

We have proposed Dynamic Policy Management Framework (DPMF) to manage policies on multiple Virtual Organizations (VOs) in a Grid. The dynamic VO membership organization of Grid computing and heterogeneous VOs are major challenges to policy management. DPMF develops an architecture which groups VOs of the same policy framework to forms virtual clusters. Each VO has a Policy Agent (PA) to store and manage the policies of the VO. Inter-"virtual cluster" heterogeneous policy management is handled by a Principal Policy Processing Unit (P-PPU). On the other

hand, intra-cluster homogeneous policy managements are handled by a Principal Processing Unit (PPU) which is one of the PAs in the cluster. A PPU learns about the trust relationships of fellow PAs to form the PA trust relationship hierarchy of the virtual cluster. By the hierarchy, the PPU can delegate policy conflict analysis tasks to other PAs, and exchange policy information.

Since the VOs have their own trust decisions, a PPU may not be trusted by all PAs in its virtual cluster. In DPMF, the Conflict Analysis with Partial Information (CAwPI) mechanism is used to generate substitution policies for unknown policies for conflict analysis. By CAwPI mechanism, PPU generates sets of policy templates by learning the authorization policies from other PAs. The policy templates store policies with the properties of the policy owners. In a conflict analysis task, if there is PA(s) not trusting the PPU for providing policy information, the PPU would deploy policy templates to generate substitution policies. The selection principle is to select policies templates with owner properties similar to the untrusting PA(s). The CAwPI mechanism would detect possible conflicts between known policies from untrusting PAs and the substitution policies. The mechanism provides an approach to analyze policy conflict in open environment without complete information.

Acknowledgements

The work described in this paper was partially supported by the following grants: RGC Competitive Earmarked Research Grants (Project ID: 2150348, RGC Ref. No.: CUHK4187/03E ; Project ID: 2150414, RGC Ref. No.: CUHK4220/04E).

References

1. Don Box, Franciso Curbera, Maryann Hondo, Chris Kale, Dave Langworthy, Anthony Nadalin, Nataraj Nagaratnam, Mark Nottingham, Claus von Riegen, John Shewchuk: Specification: Web Services Policy Framework (WSPolicy), available at http://www-106.ibm.com/developerworks/library/ws-polfram/
2. N. Dunlop, J. Indulska, K. Raymond. "Methods for conflict resolution in policy-based management systems", Proceedings of the Seventh IEEE International Conference on Enterprise Distributed Object Computing 2003, 16-19 Sept 2003, Pages 98-109.
3. I. Foster and C. Kesselman, J. Nick, and S. Tuecke: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, avaliable at http://www.globus.org, 2002. Version: 6/22/2002.
4. B. Moore, E. Ellesson, J. Strassner, A. Westerinen: Policy Core Information Model – Version 1 Specification, IETF Network Group RFC 3060, February 2001.
5. Gary N. Stone, Bert Lundy, and Geoffery G. Xie, U.S Department of Defense: Network Policy Languages: A Survey and a New Approach, in IEEE Network, Jan/Feb 2001.
6. J. Strassner and E. Ellesson: Terminology for Describing Network Policy and Services, Internet draft draft-strasner-policy-terms-01.txt, 1998.
7. Dinesh Verma, Sambit Sahu, Seraphin Calo, Manid Beigi, and Isabella Chang: A Policy Service for GRID Computing, M. Parashar(Ed.): GRID 2002, LNCS 2536, pp. 243-255.

8. Von Welch, Frank SiebenSet, Ian Foster, John Bresnahan, Karl Czajkowski, Jarek Gawor, Carl Kesselman, Sam Meder, Laura Pearlman, and Steven Tuecke: Security for Grid Services, in Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03).
9. Chiu-Man Yu and Kam-Wing Ng. "A Dynamic Management Framework for Security Policies in Open Grid Computing Environments". Proceedings of the Third International Conference of Grid and Cooperative Computing (GCC 2004), pp. 871-874. China, October, 2004.

Security Architecture for Open Collaborative Environment

Yuri Demchenko[1], Leon Gommans[1], Cees de Laat[1], Bas Oudenaarde[1], Andrew Tokmakoff[2], Martin Snijders[2], and Rene van Buuren[2]

[1] Universiteit van Amsterdam, Advanced Internet Research Group, Kruislaan 403, NL-1098 SJ Amsterdam, The Netherlands
{demch, lgommans, delaat, oudenaarde}@science.uva.nl
[2] Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
{Andrew.Tokmakoff, Martin.Snijders, Rene.vanBuuren}@telin.nl

Abstract. The paper presents proposed Security Architecture for Open Collaborative Environment (OCE) being developed in the framework of the Collaboratory.nl (CNL) project with the intent to build a flexible, customer-driven security infrastructure for open collaborative applications. The architecture is based on extended use of emerging Web Services and Grid security technologies combined with concepts from the generic Authentication Authorization and Accounting (AAA) and Role-based Access Control (RBAC) frameworks. The paper describes another proposed solution the Job-centric security model that uses a Job description as a semantic document created on the basis of the signed order (or business agreement) to provide a job-specific context for invocation of the basic OCE security services. Typical OCE use case of policy based access control is discussed in details.

1 Introduction

The process industry makes extensive use of advanced laboratory equipment, such as electron microscopes, equipment for surface analysis and mass spectrometers. Laboratories tend not to have purchased highly specialized and sophisticated equipment, due to high initial outlay and operational costs and the expertise required to operate such equipment. The Collaboratory.nl[1] project (CNL) is one of the projects that investigate how technologies for remote operation of laboratory equipment can be integrated with existing Groupware and emerging Web Services and Grid technologies for enhanced remote collaboration.

This paper presents the results of the development of an open, flexible, customer-driven security infrastructure for open collaborative applications, in particular addressing practical needs of the Collaboratory.nl project. The proposed solution is based upon extended use of emerging Web Services and Computer Grid security technologies and the generic AAA authorisation framework [1, 2, 3, 4].

[1] http://www.collaboratory.nl/

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 589 – 599, 2005.

Collaborative applications require a sophisticated, multi-dimensional security infrastructure that manages the secure operation of user applications between multiple administrative and trust domains. Typical Open Collaborative Environment (OCE) use cases imply specifics of the collaborative environment that:

- is dynamic as the environment may change from experiment to experiment,
- may span multiple trust domains,
- needs to handle different user identities and attributes that must comply with different policies that are both experiment and task specific.

Managing access based upon role, assigned privileges and policy enforcement have been addressed in many collaborative and Computer Grids projects. It can provide a good basis for the development of security architecture for OCE. The majority of known solutions and implementations use widely recognised Role-based Access Control (RBAC) [5] model and its implementation in XACML [6] as a general conceptual approach.

The current Grid Security Infrastructure and Authorisation framework evolved from using proprietary systems like Community Authorisation Service (CAS) [7] to XACML based Policy Management and Authorization Service for Grid resources [8]. Although they provide a good example of addressing similar tasks, current Grid-based solutions cannot be directly used within OCE, since their deep embedding into parallel task scheduling mechanisms prevents distributed execution of dissimilar computational tasks/jobs. A typical collaborative environment is less coupled and mostly concerned with the allocation and execution of complex experiments on the equipment that for most use cases, requires human control and interaction during the experiment.

Collaborative tools such as CHEF[2], initially designed for online educational course management, can provide most of the necessary functionality for the creation of a collaborative environment. However, this environment needs to be extended such that it can be integrated with other stages and components of the collaborative organisation managing the experiment stages. These stages include the initial stage of order creation, and the main experimental stage that requires secure access to the instrument or resource.

To address the specifics, the proposed OCE Security Architecture uses a novel Job-centric approach, which is based on Job description as a semantic document, created on the basis of a signed order (business agreement). The document contains all the information required to run the analysis, including the Job ID, assigned users and roles, and a trust/security anchor(s) in a form of customer and/or OCE provider digital signature. In general, such approach allows binding security services and policies to a particular job or resource.

The paper is organized as follows. Section 2 of the paper describes basic OCE use cases and their required security functionality. Section 3 describes the general OCE Security Architecture and its general security services model. The architecture builds upon (and is intended to be compatible with) WS-Security and OGSA Security. Section 4 describes the OCE policy enforcement framework implementation using generic AAA architecture and RBAC based Authorisation service model.

[2] http://www.chefproject.org/

Proposed solutions are being developed in coordination with ongoing projects CNL and EGEE[3], and can represent a typical use case for the general Web Services and OGSA Security framework. It is expected that other similar projects such as VL-E[4] and GridLab[5] will benefit from this work as it intends to propose a general approach and common solutions for the security problems in OCE.

2 Basic OCE Use Cases and Proposed Job-Centric Security Model

Security services are defined as a component of the OCE middleware that provides secure infrastructure and environment for executing OCE tasks/jobs. Generally, security services can be added to an already existing operational architecture, however current industry demand for very secure operational environments requires that a Security architecture is developed as an integral part of the system design. There should be also a possibility to define a security services profile at the moment of a system service invocation defined by a security policy.

For the purpose of analysis of the required security functionality, all use cases in CNL can be split into two groups of simple security interactions and extended ones. In a simple/basic use case the major task is to securely provide remote access to instrument(s) belonging to a single provider. For this case, the remote site or the resource owner can provide few onsite services and allow distributed user groups. An extended use case must additionally allow distributed multi-site services, multiple user identities and attribute providers, and distributed job execution. In its own turn, multiple trust domains will require dynamic creation of user and resource federations/associations, handling different policies, specific measures for protecting data confidentiality and user/subject privacy in potentially uncontrolled environment.
In both cases, there is a need for the following functionality:

- fine grained access control based on user/subject attributes/roles and policies defined by a resource
- privilege/attribute management by a designated person carrying responsibility for a particular experiment or job
- customer controlled security environment with the root of trust defined by the customer (in particular, their private key).

Listed above functionalities require a new job-centric approach to security services provisioning in OCE that can be realised in the following way.

Procedures in OCE include two major stages in accepting and executing the order: negotiation and signing the order (business part), and performing the experiment (technical part). The Job description, as a semantic document, is created based on the signed order and contains all information required to run the experiment on the collaborative infrastructure. The job description contains the following components: a Job ID and other attributes, Job owner, assigned users and roles, business anchor(s) (BA) and/or trust/security anchor(s) (TA) in a form of customer and provider digital signatures.

[3] http://public.eu-egee.org/
[4] http://www.vl-e.nl/
[5] http://www.gridlab.org/

Figure 1 illustrates a structure of the Job description and its relation to other OCE components and security services. This kind of job description can also be used as a foundation for creating Virtual Organisation (VO) [2] instance as an association of designated users and resources, which support all standard security constructs such as users, groups, roles, trust domains, designated services and authorities.

The job description (mandatory) must include or reference the Job policy, which defines all aspects of the user, resource and trust management when executing the job. This policy should define the following issues:

- trusted CA (Certification Authorities) or Identity Providers;
- trusted users, VO's, resources and in general trusted credentials;
- privileges/permissions assigned to roles;
- delegation policy;
- credit limits and conditions of use;
- confidentiality and privacy requirements;
- identity federation/mapping policy;
- Job access control or authorisation policy.

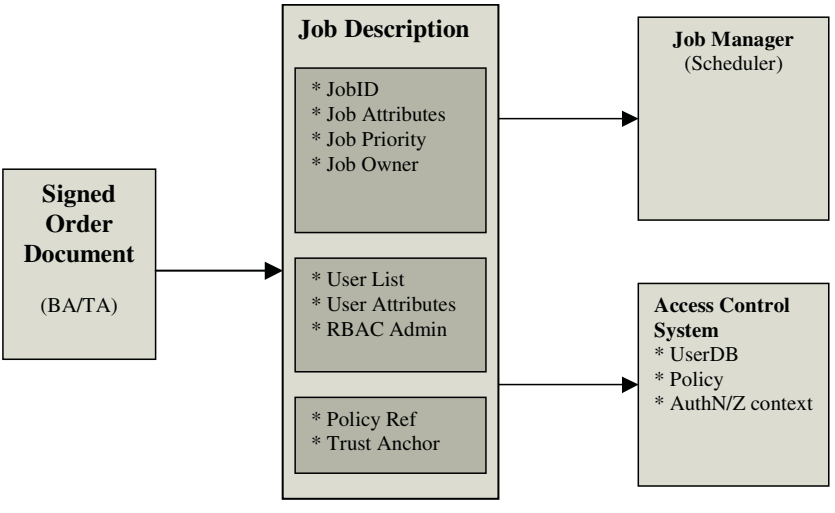


Fig. 1. OCE Security built around Job description

It is important to note that a Job policy may be combined with the Resource admission policy and in practice should not be more restrictive than the Resource policy. Otherwise the Job security management service may reject some resources based on Resource policy evaluation as a procedure of mutual authorisation.

Job-centric approach gives organizations complete flexibility in the creation of their security associations and services for the specific tasks or applications.

Practical implementation of the Job-centric security model requires wide spectrum of emerging XML and Web Services Security technologies that altogether constitute a general OCE Security Architecture as described in the next chapter.

3 Adopting Web Services Security Architecture for an Open Collaborative Environment

This section provides an overview and describes the general OCE Security Architecture based on wide use of the related WS-Security [1] and Open Grid Service Architecture (OGSA) [2] Security services and standards. The intension of this section is to provide guidance to existing XML and WS-Security standards and how they can be used for basic security services in OCE.

In Web Services Architecture (WSA) a Web Service is defined by PortTypes, Operations, Messages and Binding between all three components and actual back-end service in the form of WSDL (Web Services Description Language) description [9]. Security services and components can be added to the service description and defined by WS-Security set of standards. WS-Security components are already included into WSDP 1.4[6] and in the Globus Toolkit Version 3.2 and later[7].

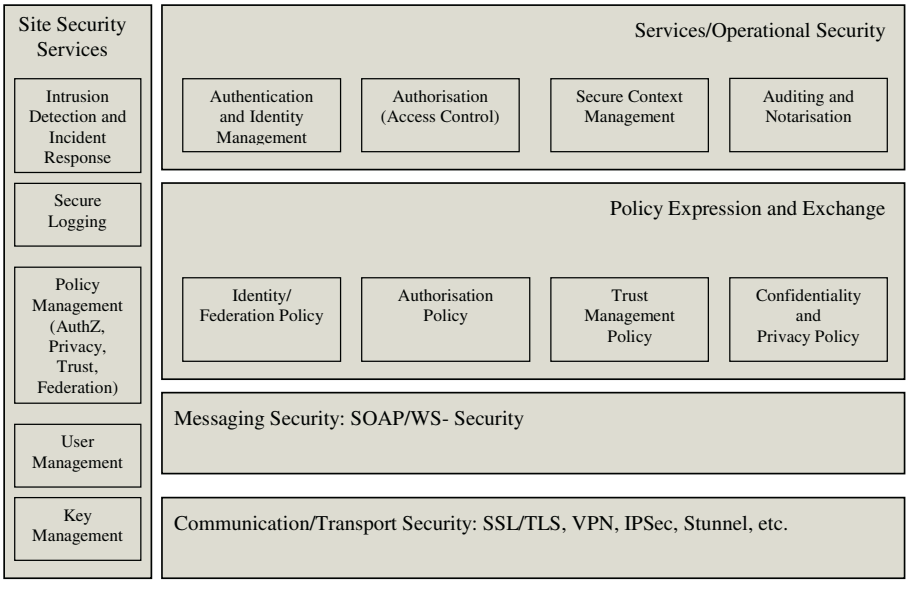


Fig. 2. Security Architecture for an Open Collaborative Environment

The OCE Security Architecture is built upon above mentioned technologies and includes the following layers and components (see Fig. 2):

1) Communication/transport Security Layer defines network infrastructure security and uses such network security services as SSL/TLS, IPSec, VPN, and others.

[6] http://java.sun.com/webservices/jwsdp/
[7] http://www-unix.globus.org/toolkit/

2) Messaging Security Layer is based on currently well-defined SOAP/WS-Security mechanisms [10] and may use SAML as a security token exchange format [11].
3) Policy Expression and Exchange Layer defines set of policies which can be applied to OCE/VO users when they interact with the environment, and which are necessary to ensure multi-domain and multiplatform compatibility.
4) Services/Operational Layer defines security services/mechanisms for secure operation of the OCE components in an open environment: authentication and identity management, authorisation and access control, trust or secure context management, auditing/logging and notarization.

Some of the layers and components important for understanding the proposed OCE security architecture and job-centric security model are described in more details below.

3.1 Policy Expression and Exchange Layer

Communicating OCE services/principals need to conform to certain requirements in order to interact securely. It is important that service or resource requestors have access to and understand policies associated with the target service. As a result, both the service requestor and service provider must select an acceptable security profile. It is also important to mention that the privilege to acquire security policy is given by the hosting environment to authenticated and authorised entities only.

The policy layer provides necessary information for the policy enforcement modules of the Operational Security layer. It is suggested that policy expression should conform to the general WS-Policy framework and may include component policies using different policy expression formats including XACML [6] or Generic AAA [12] policy formats. Policies for end applications and services are described by (a set of) rules for a specific target comprising of a triad (Subject, Resource, Action). Policy may also include common attributes such as security tokens, delegation and trust chain requirements, policy combination algorithms in a form defined by WS-Security and WS-Policy [13].

Policy association with the service can be done using WS-PolicyAttachment that defines extensions mechanisms to attach a policy or policy reference to different components of the service description including PortType, operation/message, or arbitrary element [14]. XACML Web Services profile defines the mapping between Web Service description components and its own policy description hierarchy [15].

3.2 Authentication and Identity Management

The OCE operational environment may consist of multiple locations and security domains that maintain their own Authentication and Identity management services. Interoperation and secure Single Sign-on (SSO) will require federation of the involved domains and identity and credentials translation or mapping that can be built on two currently available identity management specifications: WS-Federation [16] and Liberty Alliance Project (LAP) [17]. Preferences can be defined by other OCE

security components. LAP has benefit of being a more independent implementation, but it implements more business oriented approach. WS-Federation is more naturally integrated with Web services environments.

Authentication and Identity management services can use PKI based credentials for user/entity identification and authentication, and SAML and WS-Security for security credentials and token definition and exchange format [10, 11].

3.3 Authorisation and Access Control

The Authorisation and Access Control security service is a key part of the managed security in an open service oriented environment. Authorisation is typically associated with a service provider or resource owner. The resource owner controls access to a resource based upon requestor credentials or attributes that define the requestor's privileges or associated roles bound to the requestor's identity. Separation of Authentication and Authorisation services allows dynamic RBAC management [5] and virtual association between interacting entities, and provides a basis for privacy.

For distributed multi-domain services/applications, an Authorisation service can operate in pull or push modes as defined by the generic AAA architecture [3, 4] in which correspondently Authorisation decision is requested by a Resource service, or requestor preliminary obtains the Authorisation token from the trusted Authorisation service. It subsequently presents the token together with the authorisation context to the resource or service. When using the push or the combined pull-push model for complex services requests, the SAML format is considered as a recommended format for authorisation tokens exchange.

4 Policy Based Access Control Using RBAC Model and Generic AAA Framework

This section provides an illustration how generic security components, in particular generic AAA framework and RBAC, can be used for providing policy based access control functionality in OCE.

Security services may be bound to and requested from any basic OCE service using a standard request/response format. Security services use must be specified by the policy that provides a mapping between a request context (e.g., action requested by a particular subject on a particular resource) and resource permissions.

Binding between (core) services and security services can be defined dynamically at the moment of service deployment or invocation using existing Web services and XML Security technologies for binding/associating security services and policies to the service description as explained above.

Figure 3 illustrates basic RBAC components and their interaction during the evaluation of the authorisation request against the access policy. When combined with the Job- centric approach, the Policy components and attributes and the request context can be defined by the particular job description that binds job attributes, user information and established trust relations between the customer and the provider.

The OCE policy/role based Authorisation infrastructure consists of

- a PDP (Policy Decision Point) as a central policy based decision making point,
- a PEP (Policy Enforcement Point) providing Resource specific authorisation request/response handling and policy defined obligations execution,
- a PAP (Policy Authority Point) as a policy storage (in general, distributed),
- a PIP (Policy Information Point) providing external policy context and attributes to the PDP including subject credentials and attributes verification
- a RIP (Resource Information Point) that provides resource context
- a AA (Attribute Authority) that manages user attributes and can be in particular case VO management service (VOMS) [19].

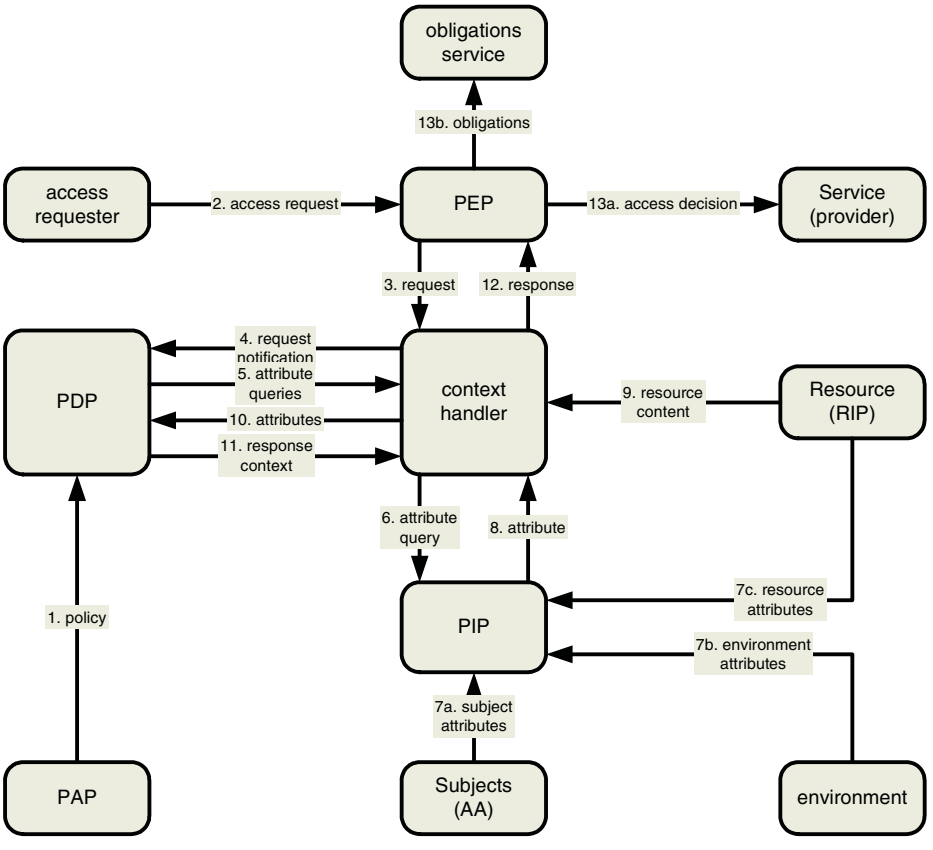


Fig. 3. RBAC based authorisation system components and dataflows

To obtain a permission to access a resource, a user or resource agent request via PEP an authorisation decision from PDP that evaluates the authorisation request against the policy defined for a particular job, resource and user attributes/roles. The access policy is normally defined by the resource owner and may be combined with

the Job policy. The PEP and PDP may also request specific user attributes or credentials from the authentication service, or additional information from the Resource.

When using XACML as a policy and messaging format, the Policy is constructed as a set of rules against the Target defined as a triad (Subject, Resource, Action), and the Request message also requests a PDP decision against a particular Target. The Subject element may contain Subject ID, Subject authentication or identity token, attributes or role(s), and other credentials. The Response message returns the PDP decision and may contain obligation conditions that must be executed by the PEP.

5 Summary and Conclusions

The general OCE Security architecture and implementation suggestions described in this paper are based on practical experience of building open collaborative environment for the Collaboratory.nl project undertaken in framework of the Collaboratory.nl consortium.

Proposed Security Architecture is based upon existing and emerging Web Services Security technologies and intends to be compatible with the OGSA Security Architecture. Existing XML and Web Services Security technologies provide a basis for building distributed Security infrastructure for OCE and easy integration with existing and developing security services and solutions. Use of industry standards such as WS-Security, XACML and SAML will guarantee future compatibility between CNL/OCE implementations and third party products for security services.

On other side, the CNL project provides a good platform for testing and further development of the proposed ideas and solutions. The CNL Security Architecture implements proposed Job-centric approach that allows building basic CNL security services around the semantic Job description document created on the base of signed order and containing all information required to run the experiment or execute the job on the CNL and enable basic security services such as user authentication, policy and role based access control, confidentiality and integrity of information and data.

The CNL Authorisation framework combines Web Services security mechanisms with the flexibility of the Generic AAA Architecture and XACML policy/role based access control model to built fine-grained access control. Separating policy definition from the authorisation or access control execution/enforcement will simplify access control management, which can be delegated to the resource owner. To reduce performance overhead when requesting authorisation decision from PDP, CNL implementation combines pull and push models [4] by using authorisation ticket with the limited validity period that allows to bypass potentially slow request evaluation by PDP.

The CNL project is being developed in coordination with the EGEE project what will allow future use of the Grid infrastructure being development in the framework of EGEE project and guarantee the compatibility of basic security services such as authentication, authorisation, and corresponding formats of metadata, policies, messages, etc.

Authors believe that the proposed OCE Security architecture and related technical solutions will be interesting for other projects dealing with the development of middleware for virtual laboratories and collaborative applications. Discussed here OCE specific components will be available openly as a part of the GAAA Toolkits by the middle of 2005.

Acknowledgements

This paper results from the Collaboratory.nl project, a research initiative that explores the possibilities of remote control and use of advanced lab facilities in a distributed and collaborative industrial research setting. The Collaboratory.nl consortium consists of DSM, Philips, Corus, FEI, Telematica Instituut and the University of Amsterdam.

References

1. Security in a Web Services World: A Proposed Architecture and Roadmap, Version 1.0, A joint security whitepaper from IBM Corporation and Microsoft Corporation. April 7, 2002, http://www-106.ibm.com/developerworks/library/ws-secmap/
2. The Open Grid Services Architecture, Version 1.0 – 12 July 2004, 2004. - http://www.gridforum.org/Meetings/GGF12/Documents/draft-ggf-ogsa-specv1.pdf
3. RFC 2903 , Experimental, "Generic AAA Architecture",. de Laat, G. Gross, L. Gommans, J. Vollbrecht, D. Spence, August 2000 - ftp://ftp.isi.edu/in-notes/rfc2903.txt
4. RFC 2904 , Informational, "AAA Authorization Framework" J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, D. Spence, August 2000 - ftp://ftp.isi.edu/in-notes/rfc2904.txt
5. Role Based Access Control (RBAC) – NIST, April 2003. - http://csrc.nist.gov/rbac/
6. eXtensible Access Control Markup Language (XACML) Version 1.0 - OASIS Standard, Feb. 2003 - http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0. pdf
7. K.Keahey, V.Welch, “Fine-Grain Authorization for Resource Management in the Grid Environment”. - http://www.fusiongrid.org/research/papers/grid2002.pdf
8. Markus Lorch, Dennis Kafura, Sumit Shah, "An XACML-based Policy Management and Authorization Service for Globus Resources". - Grid 2003, 17 November 2003. - http://zuni.cs.vt.edu/publications/grid-authz-policy-mgmt-wip03.ps
9. Web Services Architecture, W3C Working Draft 8 August 2003 - http://www.w3.org/TR/ws-arch/
10. Web Services Security Framework by OASIS - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
11. Security Assertion Markup Language (SAML) v1.0 - OASIS Standard, Nov. 2002 - http://www.oasis-open.org/committees/documents.php?wg_abbrev=security
12. A grammar for Policies in a Generic AAA Environment - http://www.ietf.org/internet-drafts/draft-irtf-aaaarch-generic-policy-03.txt
13. Web Services Policy Framework (WS-Policy). Version 1.1. http://msdn.microsoft.com/ws/ 2002/12/Policy/
14. Web Services Policy Attachment (WS-PolicyAttachment). Version 1.1. - http://msdn.microsoft. com/ws/2002/12/PolicyAttachment/

15. XACML profile for Web-services (WSPL): - http://www.oasis-open.org/committees/download.php/ 3661/draft-xacml-wspl-04.pdf
16. Web Services Federation Language (WS-Federation) Version 1.0 - July 8 2003 – http://msdn.microsoft.com/ws/2003/07/ws-federation/
17. Liberty Alliance Phase 2 Final Specifications - http://www.projectliberty.org/specs/
18. Demchenko Yu. Virtual Organisations in Computer Grids and Identity Management. – Elsevier Information Security Technical Report - Volume 9, Issue 1, January-March 2004, Pages 59-76.

An Experimental Information Grid Environment for Cultural Heritage Knowledge Sharing

A. Aiello, M. Mango Furnari, and A. Massarotti

Istituto di Cibernetica E. Caianiello, Via Campi Flegrei, 34,
I-80078 – Pozzuoli, Italy
{a.aiello, mf, a.massarotti}@cib.na.cnr.it

Abstract. In this paper the authors address the problems of making existing distributed collection document repositories mutually interoperable at the semantic level. The authors argue that semantic web technologies offer a promising approach to facilitate homogeneous, semantic information retrieval based on heterogeneous document repositories on the web. From contents point of view, the distributed system is built as a collection of multimedia documents repository nodes glued together by an ontology server. A set of methodologies and tools for organizing the information space around the notion of contents community is developed, where each content provider will publish a set of ontologies to collect metadata information organized and published through the Contents Community Authority on top of an ontology server. These methodologies were deployed setting up a prototype to connect about 20 museums in the city of Naples (Italy).

1 Introduction

In this paper the authors address the problem of making distributed document collection repositories mutually interoperable at semantic level. Furthermore, they argue that emerging semantic web technologies, more specifically the ontology one, offer a promising approach to facilitate semantic information retrieval based on heterogeneous document repositories distributed on the web. However, the current source ontologies exploitation attempts are oriented to cope with the conceptualization of single information source. Furthermore, most of existing tools treat ontologies as monolithic entities and provide little support for specifying, storing and accessing ontologies in a modular manner.

The authors' efforts described in this paper are based on the hypothesis that it is necessary to develop an adequate treatment of distributed ontologies in order to promote information sharing on the semantic web, and appropriate infrastructures for representing and managing distributed ontologies have also to be developed. To pursue these goals we defined a modularized ontology representation and developed an ontology server to deploy a knowledge repository community. An experimental implementation to verify the developed methodologies and tools within the cultural heritage promotion arena is also described.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 600–609, 2005.

The rest of the paper is organized as follows: In the first section the architecture and the implementation of the proposed Distributed Contents Management System are given together the Ontology Server architecture. In the second section a modular representation for the ontology structure is described. In the third section the implemented test bed is described. In last section the proposed architecture advantages are summarized and compared with other efforts.

2 The Distributed Contents Management System and Ontology Server Architecture

We chose the WWW paradigm as design criteria for a distributed contents management system, where the notion of *document* plays the role of elementary information and basic building block. Documents are represented as digital objects together with the associated metadata information, where the metadata are organized using domain ontology. Furthermore, we assumed the multi-tiers web architecture, with the application server playing the central role of business logic driver, where the main identified components are:

- *Document Repository System* (*DRS*). The DRS stores and organizes the documents together with the associated metadata.
- *Document Access System* (*DAS*). The DAS creates friendly and flexible user interfaces to discover and access the contents.
- *Contents Authority Management System* (*CAS*). The CAS stores and manages the ontologies used by each participating node to facilitate the DRS semantic interoperability.

All these systems communicate among them exchanging XML encoded messages over http, according to well-defined protocols that represent the XML communication bus core, see Figure 1.

The user will interact with the community of systems through a conventional browser; the DRS appears and behaves like a traditional web site. Documents must underway a text processing before to be displayed, and programmed according to a sequence of transformations expressed using the eXtensible Stylesheet Language Transformation (XSLT) [7]. The Document Access System manages this document composition process, whose business logic could be summarized as follows: the ontology client makes the first step by extracting the information from the data store and wrapping this information with XML tags. The extraction is done querying the ontology server. The second step involves the application of the appropriate stylesheet transformations to the XML data and thereby the creation of a corresponding HTML page. The foregoing step is carried out by the XSLT package included in the application server. The output of that transformation is the HTML page directly sent to the browser.

The advantages of the whole proposed architecture are: a) ease of deployment on Internet, high reliability and fault-tolerance, and efficient use of the network infrastructures; b) flexibility and generality as needed in order to evolve and

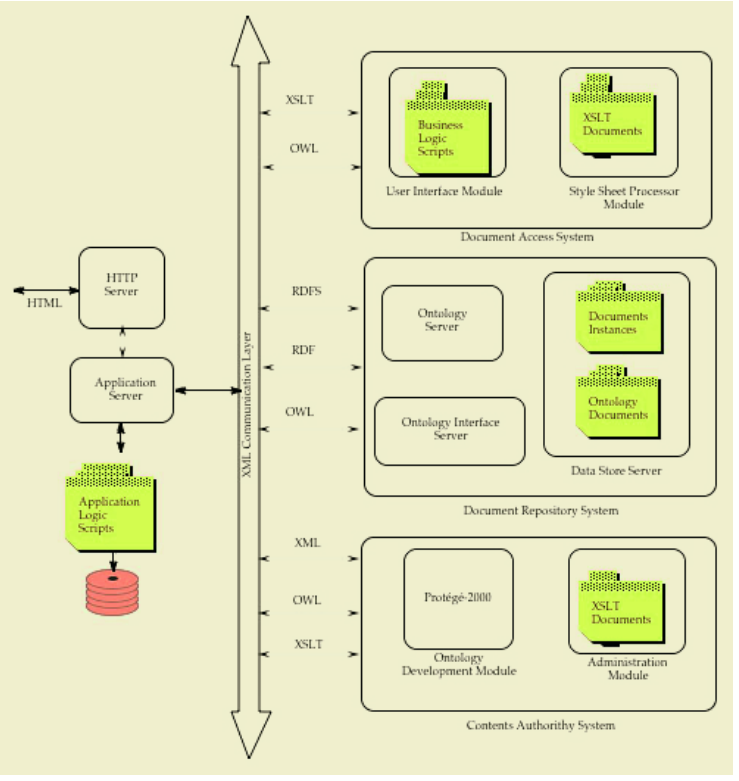


Fig. 1. Distributed Contents Management architecture

meet future needs; c) scalability without fundamental changes in the structure of the resource name spaces.

In the previous scenario the ontology server provides the basic semantic interoperability capabilities to build a Document Repositories Community. From the conceptual point of view the Ontology Server it is the most important type of servers since it manages the OWL/RDF [15] schema for the stored data, and determines the interactions with the other servers and/or modules, through the ontology exchange protocol [13].

The Ontology Server provides the information provider with the possibility of interacting with heterogeneous and distributed document repositories. Actually, it guarantees the necessary autonomy to the information provider in organizing their contents space. To achieve these goals the Ontology Server is equipped with the following modules:

– *Ontology Development Module.* The ontology development module is built around the Protégé-2000 [19] ontology editor, since its architecture is modular and extensible. We developed an extension for the OWL Protégé-2000 Plug-in in order to store the ontology directly on the Data Store Module using the client/server metaphor, see Figure 2.

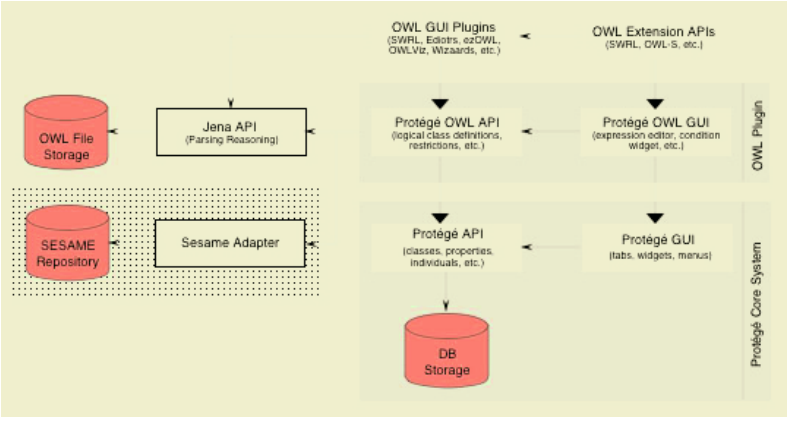


Fig. 2. The Plugin OWL and Tab architecture

- *Ontology Repository Module.* For the OWL/RDF data persistent storage we choose the Sesame package [3]. It is an open source, platform-independent, RDF Schema-based repository, provided with querying facility written in Java. The low level persistent storage is achieved using Postgresql [18], one of most widely used public domain database environment. The Sesame environment offers three different levels of programming interfaces: the client API, for client-server programming; the server API; and the lower level *Storage and Inference Layer* (SAIL) API, for the RDF repositories.
- *Ontology Interface Module.* The Ontology Interface Module consists of a set of functionalities for walking through the ontology graph and the associated attributes. At runtime, these functionalities could be used by a Document Access System to build the user interfaces, to browse the ontology structures, to implement an ontology driven search engine, and so forth. The Ontology Interface Module can be queried about the ontology class hierarchy, and/or the class properties, giving back an RDF document that could be transformed into HTML forms.

3 Ontology Modular Representation

The main purpose of building an ontology is to capture the semantics of the documents describing a given knowledge domain, especially the conceptual aspects and interrelations. We used OWL DL [17] to represent domain of concepts and relationships in a machines and humans understandable form. OWL DL is a rich ontology language with built-in semantics. It allows exploiting the well-defined semantics of description logics, where a reasonable upper bound is guaranteed for the complexity inconsistency, misclassifications, misunderstandings and deductions checking.

To cope with the interoperability problems related to exchange ontologies among cooperating information systems, we took into account the fact that interoperability worst case occurs when *useful* communication is restricted to transfer an ontology, as a whole such as happens with the currently serializing language and/or schema. By contrast, we may expect that transferring ontology in small *meaningful* chunks could significantly improve the knowledge system interoperability.

We defined a language for the XML serialization of the OWL DL ontologies, called *ezXML4OWL* [14]. The idea was to serialize an OWL mereology by mapping whole OWL ontologies to whole XML documents as well as parts belonging to the OWL mereology to the corresponding XML elements, all of them with the constraint that the relation *part-of* corresponds to the relation *XML-element-of*. Of course, it is not required that every XML-element occurring in ezXML4OWL have a correspondent in the OWL mereology. There are, indeed, auxiliary XML elements that have only serializing roles. Moreover, some redundancy was created to make our representation more understandable, and to make ezXML4OWL documents modular.

An ezXML4OWL document serializing an OWL DL ontology is composed of exactly the following three modules[1]: `<ontology>`, `<axioms>`, and `<facts>`. These modules are the top modules and like other modules and/or elements are recursively defined. The module `<ontology>` encodes metadata about the ontology, such as the name and the four OWL built-in ontology properties: `owl:imports`, `owl:priorVersion`, `owl:backwordCompatibleWith` and `owl:incompatibleWith`. The ontology's name and the associated properties are encoded according to the following skeleton:

```
<ontology name=ontologyID>
      <ontoProperty name= owl:priorVersion>
         <value name= ontologyID>
      </ontoProperty>
      <ontoProperty name= owl:backwordCompatibleWith>
         <value name= ontologyID>
         .....
         <value name= ontologyID>
      </ontoProperty>
      <ontoProperty name= owl:incompatibleWith>
         <value name= ontologyID>
         .....
         <value name= ontologyID>
      </ontoProperty>
      <ontoProperty name= owl:imports>
```

[1] We use the term "module" for referring to ezXML4OWL elements that codify concepts belonging to the OWL mereology. Henceforth, the term "element" will refer to generic XML elements (not necessarily modules).

```
         <value name= ontologyID>
         .....
         <value name= ontologyID>
      </ontoProperty>
</ontology>
```

The module <facts> stores all the data gathered during the data entry phase. Since <facts> modules are about individuals, they might contain only modules of the type <individual>. Anyway, since each <individual> module would codify an instance of a class we defined a module, of the type <classIndividuals>, corresponding to the mereological entity "set of instances in the same class", and operating as a container for all <individual> codifying instances in the same class. <classIndividuals>. The fact module skeleton is:

```
<facts>
   <classIndividuals className>
      <individual name>
          <individualID_ValuedProperties>
               <property name >
                   <value name/>
                   ......
               </property>
               .....
               .....
          </individualID_ValuedProperties>
          <data_ValuedProperties>
                <property name >
                     <value name/>
                </property>
                .....
          </data_ValuedProperties>
      </individual>
      .....
</classIndividuals>\\
.....
.....
</facts>
```

We codify the different kind of ontology axioms in different modules, all being direct parts of the module <axioms>, where the module <classesLattice> explicitly describes the lattice formed by the classes associated to the ontology to be serialized. The lattice's structure is specified giving the direct subclasses of each class. The module <classesLattice> implements both the first two types of description prescribed by OWL: the class identifier (a URI reference) and the property rdfs:subClassOf. The module <classesSlots> codifies the classes and the related properties. There are modules for each type of OWL property.

Namely, the `owl:DatatypeProperty`'s are encoded in modules `<dataProp/>` and the `owl:ObjectProperty` are encoded in modules `<obProp>`. Attributes are also given to specify the range, the cardinality and the source (inherited or specific) of the properties, the remaining modules description and remarks can be found in [17].

The axioms module skeleton is:

```
<axioms>
   <classesLattice>
      <root name ‘‘classeID’’/>
      .....
      <root name =‘‘classeID’’/>
      <leaf name =‘‘classeID’’/>
      .....
      <leaf  name=‘‘classeID’’/>
      ....
      <sup name =‘‘classeID’’>
          <sub name =‘‘classeID’’/>
          .....
          <sub name =‘‘classeID’’/>
      </sup>
    </classesLattice>
    <classesSlots>
        <class name ‘‘classeID’’> ..... </class>
        ......
        <class name ‘‘classeID’’> ..... </class>
        .....
    </classesSlots>
    <subClassOf>          ..... </subClassOf>
    <enumeratedClasses>   ..... </enumeratedClasses>
    <equivalentClasses>   ..... </equivalentClasses>
    <disjointClasses>     ..... </disjointClasses>
    <objectProperties>    ..... </objectProperties>
    <datatypeProperties> ..... </datatypeProperties>
</axioms>
```

4 The Museo Virtuale di Napoli Testbed

The aim of any ordinary museum visitor is something quite different from trying to find certain objects. In physical exhibitions, the cognitive museum experience is often based on the thematic combination of exhibits and their contextual information. In order to figure out how much it would be complex to achieve these goals and which kind of technologies would be necessary, the research

project "Museo Virtuale di Napoli: Rete dei Musei Napoletani" (REMUNA)[2] is carried out at the Istituto di Cibernetica E. Caianiello. The collection of eighteen Neapolitan museums document repositories are used as case study. These repositories use different technologies, have different conceptual schemas and are physically located in different districts of Naples.

Each museum is equipped with multimedia information system and communication infrastructures. From the museum managers' perspective each information system allows him to make available the managed artifacts' information through the ReMuNa environment, just after registering them into the system. This information is encapsulated into a digital object that plays the role of a handle for the actual artifact information. No assumption about fixed attributes names' schemata is taken, so the application builder can create new attributes, as needed just modifying the associated ontology without changing the internal database schemata.

The information provider[3] could also organize a set of related documents, in document collections, according to some relationships defined on top of the associated ontology. The adopted notion of collection is a recursive one, in the sense that a collection could contain other collections. Each digital document is allowed to belong to multiple collections and may have multiple relationships with other documents. This nesting features are represented by the document repository collection graph, and allows the system to deliver more than one logical view of a given digital documents asset.

To assure the necessary operational autonomy to the museum manager, without reducing the cooperation opportunities with other museum managers, we deployed this cooperation schema as an intermediate coordination organization that is in charge to register, syndicate and to guarantee the document contents quality, that we called Content Authority. The presence of the content authority could create a bottleneck; therefore the notion of delegation was introduced. In other words, the top authority could delegate another organization to operate as Cultural Heritage Contents Authority, on its behalf, for a more specific knowledge domain.

The domain ontology developed to exchange cultural heritage data has many common features with the CRM/CIDOC [9] have been developed. The designed cultural heritage ontology is empirical and descriptive one; it formalizes the semantics necessary to express stated observations about the world in the domain of museum documentation. It is composed of a class hierarchy, named classes interlinked by named properties. It follows object oriented design principle, the classes in the hierarchy inherit properties from their parents. Property inheritance means that both classes and properties can be optionally sub–typed for

[2] The project "Museo Virtuale di Napoli: Rete dei Musei Napoletani" is supported by Ministero dell'Università, Ricerca e Tecnologia, under contract C29/P12/M03, from here on denoted with ReMuNa.

[3] In this paper we assume that *museum manager* means the responsible, inside the museum organization, of the cultural heritage goods information.

specific applications, making the ontology highly extensible without reducing the overall semantic coherence and integrity.

The ontology is expressed according to the OWL semantic model, this choice yelds a number of significant benefits, for example the class hierarchy enables us to coherently integrate related information from different sources at varying levels of detail.

5 Conclusions

One of the most interesting technological aspects investigated was how to design document repositories systems that allow the museum manager to organize the cultural heritage heterogeneous information space spread in many autonomous organizations.

Ontology Exchange Protocol and tools were implemented to exploit the Multimedia Document Information System federation settlement. The ontology exchange protocol is very similar to the Dienst [12] collection service, where the main difference relies on the fact that in our case the collections are entities built on top of a domain ontology describing the domain of the documents content and not predefined ones. To a certain degree, our usage is similar to that of the CIMI project [4]. In fact, it has become increasingly evident that simple application-specific standard, such as Dublin Core (DC) [5], cannot satisfy the requirements of communities such as BIBLINK [2] and OAI [16] that need to combine metadata standards for simple resource discovery process.

Our work successfully showed that an RDF data store (Sesame) could be used as a backend document repository for a distributed Contents Management System (CMS), and the central role that the Ontology Server plays on deploying such kind of systems.

As the Semantic Web begins to fully take shape, this type of distributed CMS implementation will enable agents to understand what is actually being presented in distributed CMS, since all content within the system is modeled in machine understandable OWL/RDF.

Starting from these encouraging results we are planning to actively pursue some of the goals foreseen by the Semantic Web Initiative [1], [10], [11]. For example, to gain more semantic information we are exploiting pieces of well-known and supported ontologies, like ICOM-CIDOC [9].

Acknowledgment

Acknowledgments are expressed to all the people of Istituto di Cibernetica E. Caianiello that worked on the ReMuNa project, for their help, and fruitful discussions, and also to all the staff members of the Soprintendenza ai Beni Archeologici delle Province di Napoli e Caserta, Soprintendenza ai beni Artistici, Storici e Demo Antropologici della Provincia di Napoli, Soprintenda ai Beni Architettonici ed Ambientali della Provincia di Napoli, Archivio di Stato di Napoli, to

the people of Direzione Musei of Comune di Napoli, and the Assessorato alla Cultura of Comune di Napoli, without their assistance the REMUNA project and activities would not exist.

References

1. Berners-Lee, T., "WWW: Past, Present, and Future", IEEE Computer, **29**, (1996)
2. "The BIBLINK Core Application Profile", http://www.schemas-forum.org/registry/biblink/BC-schema.html
3. Broekstra J., Kampman A., van Harmelen F., "Sesame: A generic architecture for storing a querying rdf and rdf schema", In *The Semantic Web – ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pp. 54-68 (2002)
4. "CIMI: Consortium of Museum Intelligence", http://www.cimi.org/
5. "The Dublin Core Metadata Initiative", http://www.purl.org/dc/
6. Davis J. and Lagoze C., "The Networked Computer Science Technical Report Library", Cornell CS TR96-1595
7. "Extensible Style Language for Transformation", http://www.w3c.org/Style/XSLT
8. Lassila O., Swick R., "Resource Description Framework (RDF) Model and Syntax", World Wide Consortium Working Draft
9. "ICOM/CIDOC Documentation Standard Group, Revised Definition of the CIDOC Conceptual Reference Model", 1999, http://cidoc.ics.forth.gr/
10. HP Labs Semantic Web Research, "Jena-A Semantic Web Framework for Java", 2004 http://www.hpl.hp.com/seweb/
11. Horrocks I., Tessaris S., "Querying the Semantic Web: a Formal Approach". The 1^{st} International Semantic Web Conference (ISWC2002), Sardinia, Italy, June 9-12, 2002
12. Lagoze C., Shaw E., Davis J.R. and Krafft D.B., "Dienst: Implementation Reference Manual", May 5, 1995.
13. Mango Furnari M., Aiello A., Caputo V. Barone V., "Ontology Server Protocol Specification", ICIB TR-12/03
14. Mango Furnari M., Aiello A., Massarotti A., "ezXML4OWL: an easy XML for OWL", ICIB TR-06/04.
15. McGuinness D., van Harmelen F. (eds)., "OWL Web Ontology Language Overview", 2003 http://www.w3.org/TR/2003/WD-owl-features-20030331/
16. "Open Archives Initiative", http://www.openarchives.org
17. "OWL Web Ontology Language Overview", http://www.w3.org/TR/2003/PR-owl-features-20031215/
18. http://www.postgresql.org/
19. http://protege.stanford.edu

Implementation of Federated Databases Through Updatable Views

Hanna Kozankiewicz[1], Krzysztof Stencel[2], and Kazimierz Subieta[1,3]

[1] Institute of Computer Sciences of the Polish Academy of Sciences, Warsaw, Poland
{hanka, subieta}@ipipan.waw.pl
[2] Institute of Informatics Warsaw University, Warsaw, Poland
stencel@mimuw.edu.pl
[3] Polish-Japanese Institute of Information Technology, Warsaw, Poland

Abstract. We present a new approach to the grid technology that is based on federated databases and updatable views. Views are used in two ways: (1) as wrappers of local servers that adopt local schemata to the federated database requirements; (2) as a facility for data integration and transformation into a canonical form according to the federated database schema. Views deliver virtual updatable objects to global clients. These objects can be associated with methods that present the procedural part of remote services, like in Web Services. The fundamental quality of the approach is transparency of servers: the user perceives the distributed environment of objects and services as an integrated virtual whole. The approach is based on a very simple and universal architecture and on the stack-based approach, which treats query languages as a kind of programming language.

1 Introduction

Grid technology provides a new information processing culture that integrates many local services into a big virtual service, summing the resources belonging to particular services. The grid user is interested in services rather than in service providers. For instance, if the grid integrates thousands of small bookstores, the customer is looking for a cheapest required book and can buy it without interesting in concrete bookstores offering this book (see Amazon's Z-shops). The service providers transparency has to be supported by many technical forms of transparency that facilitate designers and programmers of grid applications.

In this paper we focus on data-intensive applications of the grid technology where distribution of data implies distributed and parallel computing. From this point of view, the technology can be perceived as continuation of distributed/federated databases – the topic that has been developed for many years. The domain of federated databases worked out many concepts that are very close to the current grid research, such as various forms of transparency, heterogeneity, canonical data models, transaction procession within distributed federations, metamodels for federations, etc. In our opinion the concepts, ideas and solutions worked out by the distributed/federated databases domain must sooner or later be absorbed by the field of grid technology.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 610 – 619, 2005.

The key issue behind data integration in grid is *transparency* i.e., abstraction from secondary features of distributed resources. There are many forms of transparency like location, access, concurrency, implementation, scaling, fragmentation, replication, indexing, or failure transparency. Due to transparency (implemented at the middleware level) some complex features of distributed data/service environment need not to be taken into account in the application code. Thus, transparency much amplifies programmers' productivity and supports flexibility and maintainability of software.

In this paper we present a new approach to implementation of several forms of transparency based on universal architecture for integration of local, autonomous databases. Data and services of a particular server are made visible for global applications through a *wrapper* that virtually maps the data/services to some assumed *canonical object model*. Then, all contributing services are integrated into the virtual whole by means of a *updatable views*. In this way all participating local data and services are seen through a single federated schema.

The novelty of the presented approach is that we propose to use updatable object-oriented views with full computational power. Views are defined in the query language SBQL that is integrated with imperative constructs (e.g. updating) and abstractions (functions, methods, procedures). The idea of using views for integration of distributed/federated databases is not new (see e.g. [Bell97, Subi00, Hal01]) and there are some prototype implementations [KLN+04, KW96]. However, to the best of our knowledge the issue is still challenging because of updatability and practical universality of view definitions. In our recent research [KLS03] we have developed and implemented object-oriented virtual views that have full algorithmic power and are updatable with no anomalies and limitations. Our views support full transparency of virtual objects, i.e. the programmer is unable to distinguish stored and virtual objects. Such views are considered as a general facility for integrating distributed and heterogeneous resources, including methods acting on virtual objects. Views deliver virtual data to the clients of a federation. Therefore, such views offer similar facilities as Web Services or a middleware based on CORBA. The advantage of our approach is that it offers a very simple architecture that is much more flexible and universal than the technologies mentioned above.

The rest of the paper is structured as follows. In Section 2 we describe Stack-Based Approach and its query language SBQL. In Section 3 we sketch the approach to updatable views. In Section 4 we explain the ideas of our grid approach. In Section 5 we present an example application of the approach. Section 6 concludes.

2 Stack-Based Approach

Our grid mechanism is based on the Stack-Based Approach (SBA) to query languages [SKL95]. SBA aims at integrating the concepts of programming languages and query languages. Therefore, queries are evaluated using mechanisms that are common in programming languages, such as stacks.

2.1 SBA Data Store

In SBA each object has the following properties: a unique internal identifier, an external name, and a value. Basing on the value we distinguish three kinds of objects:

- atomic object where the value is an atomic value (number, string, blob, etc.);
- link object with the value being a pointer to another object;
- complex object where the value is a set of objects (the definition is recursive and allows one to model nested objects with no limitations on nesting levels).

An SBA store consists of: the structure of objects/subobjects (as defined above), identifiers of root objects (starting points for queries), and constraints.

2.2 Environment Stack and Name Binding

SBA is based on the programming languages' naming-scoping-binding principle. Each name occurring in a query/program is bound to a proper run-time database/program entity according to the name scope. Scopes for names are managed by means of an Environment Stack (ES). The stack supports the abstraction principle what means that a programmer can consider a piece of code to be independent of the context of its use.

ES consists of sections, which contain entities called binders. Binders relate names with run-time objects and are used during binding names. A binder is a pair (n, i), written as $n(i)$, where n is an object name, and i is an object identifier.
New sections on ES are built by means of a special function *nested* that works in the following way:

- For the identifier of a link object the function returns the binder of the object the link points to.
- For a binder the function returns that binder.
- For a complex object the function returns binders its sub-objects.
- For structures $nested(\text{struct}\{x_1, x_2,...\}) = nested(x_1) \cup nested(x_2) \cup ...$
- For other elements the result of *nested* is empty.

The process of name binding is the following. When the query interpreter wants to bind name n occurring in a query it searches ES for a binder with the name n that is the closest to the top of ES. The process of binding respects scoping rules what means that some sections of ES can be not visible during the binding. The name binding can return multiple binders and this way we can handle collections.

In this paper w present examples basing on a simple database that keeps information on Books in a form Book(title, author, ISBN, price). Example ES states during evaluation of a query *Book* **where** *author* **=** "Niklaus Wirth" are illustrated in Fig. 1. In the initial and final states the ES contains only base section with binders to root objects. During evaluation the query the operator **where** puts on ES a new section with the environment (internal objects) of the particular Book object. This section is created by means of the function *nested* which for a book i_d returns binders: $author(i_w)$, $title(i_x)$, $price(i_y)$, $ISBN(i_z)$. After the query evaluation ES returns to the previous state.

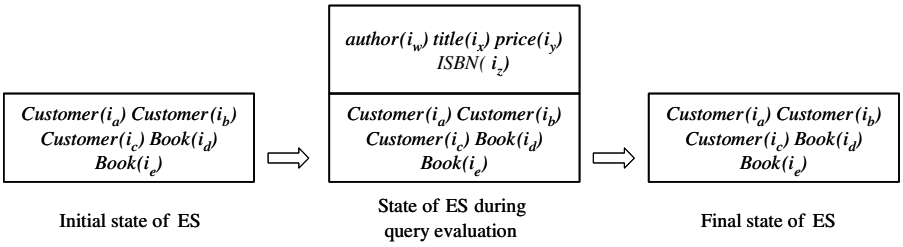


Fig. 1. States of ES during evaluation of query *Book* ***where*** *author* **=** "Niklaus Wirth"

2.3 Stack-Based Query Language

SBA has its own query language – Stack-Based Query Language [SKL95] that is based on the principle of compositionality what means that more complex queries can be built of simpler ones. In SBQL queries are defined as follows:

1. A name or a literal is a query; e.g., 2, "Niklaus Wirth", *Book*, *author*
2. σ q, where σ is an unary operator and q is a query, is a query; e.g., *count*(*Book*), *sin*(*x*).
3. q_1 τ q_2, where τ is a binary operator, is a query; e.g., 2+2, *Book.title*, *Customer* **where** <condition>

Due to the compositionality discipline that recursively defines semantics of complex queries on semantics of its components, up to primitive queries (literals and names), the whole semantics of SBQL is precise, minimal, easy to implement and optimize, and easy to understand by the users.

SBQL can be extended to support procedures. They can be defined with or without parameters, can have local environment, can have side-effects, and can call other procedures (including recursive calls).

3 View Mechanism

A view is a mapping of stored data into virtual ones. In the classical approach (SQL) a view is a function, similar to programming languages' functions. View updating means that a function result is treated as an l-value in updating statements. Such an approach, however, appeared to be inconsistent due to problems with finding unequivocal mapping of updates on virtual data into updates on stored data. In our approach to view updates [KLS03] a view definer can include into a view definition information on update intents for the given view. It allows us to eliminate ambiguities connected with multiple ways of performing a view update and the risk of warping user intention, which is a well-known problem.

Therefore, in our approach a view definition consists of two main parts. The first part describes a mapping between stored and virtual objects. The second part describes operations that can be performed on virtual objects. A view definition can contain other elements like definition of subviews, procedures, internal state variables, etc.

The first part is a procedure that returns entities called *seeds* that ambiguously identify virtual objects. Seeds are also parameters for the procedures describing operations on virtual objects that are defined in the second part of the view definition. In the approach we identified four generic operations on virtual objects:

- Deletion of a given virtual object.
- Update that changes value of a given virtual object. The operation has a parameter that is a new value of the object.
- Insertion of a new object into a given virtual object. The operation has a parameter that is an object to be inserted into the given virtual object.
- Dereference that returns a value of the given virtual object.

If an operation is not defined, it is forbidden. Internally, each operation on a virtual object has a fixed name – respectively *on_delete*, *on_update*, *on_insert*, and *on_retrieve*. The names of parameters of operations *on_update* and *on_insert* can be freely chosen by the view definer. The name of a view definition is different from the name of virtual objects – we introduced a naming convention where name of the view definition is written with a suffix "Def".

Here, we present an example definition of a view returning books that have never been bought in a bookstore:

```
create view worstSellingBookTitleDef {
        virtual objects worstSellingBookTitle {
                                return (Books where count( bought_by ) = 0) as b; }
        on_retrieve do { return b . title;}
        on_update new_title do { b . title := new_title; }
}
```

The view defines the following operations: dereference that returns title of the book, and update that changes the title of the book to the new value. We can call the view:

```
( worstSellingBookTitle as wsb  where wsb = "XMl" ) :=  "XML"
```

Mind, that in the query we call operation *on_retrieve* when comparison "=" is performed and operation *on_update* when ":=" is performed.

In our approach views can be nested, with no limitations concerning number of nesting levels. We follow the principle of relativity what means that at each level of nesting hierarchy a view is defined using the same syntax, semantics, and pragmatic. Subviews of the given view are seen as subobjects of a given virtual object.

4 Novel Approach to Integration of Federated Databases

There are many aspects of the grid technology that have to be considered for real applications, in particular, transparency, security, interoperability, efficiency, computational universality, etc. Here we concentrate only on the transparency and computational universality aspects, presenting a general architecture of a grid application and some discussion concerning adaptation of SBA to this architecture.

4.1 Architecture

The heart of our approach to federated databases is *the global virtual object and service store* (we will use also a shorter term i.e. the *global virtual store*). It stores addresses of local servers and has some computation power. It involves a query engine with ES. *Global clients* are applications that send requests and queries to the global virtual store. Global virtual store is stateless thus can be replicated by many clients.

The *global schema* is a collection of definitions of data and services provided by the global virtual store. Application programmers are the main users of these definitions (also the grid organizer uses it; see below). The programmers make use of them, while they are creating global clients. The global schema is agreed upon by a consortium (of course it may be a single company) that funds the creation of the grid.

The grid offers services and data of *local servers* to these global clients. The *local schema* defines data and services inside a local server. The syntax and semantics of these schemata as well as the natures of the data and services can be very distinct at each local server. They can be written in e.g. OMG IDL, WSDL and ODL. However, this schema is invisible to the grid.

The first step of the integration of a local server into the grid is done by the administrator of this local server. She has to define the *contributory schema* which must conform to the global schema (we will describe this conformance later in this paper). It is the description of the data and services contributed by the local server to the grid. The local server's administrator also defines *contributory views* that constitute the mapping of the data and of the local server to the data and services digestible to the grid.

The second step of the integration of local servers into the grid is the creation of *global views*. These views are stored inside the global virtual store. The interface of them is defined by the global schema. They map the data and services provided by the local servers (in the contributory schema) to the data and services available to the global clients (in the global schema).

Such a grid would be useless if we had not had updatable views. The global clients not only query the global virtual store but also request its updates. The same concerns the grid that not only queries the contributory views but also updates data of it.

Note that the view in SBA are just complex objects, thus they can contain methods and procedures as well as the local variables that store the state of these views. Therefore these views can offer the same facilities as CORBA or Web Services. There is no limitations, because SBQL has the power of universal programming languages.

The *communication protocol* is the collection of routines used in the definition of the global views. It contains the functions to check e.g. the state (up or down) of a local server and the access time to a local server.

The global views are defined by the *grid designer*, which is a person, a team or software that generates these views upon the contributory schemata, the global schema and the integration schema. The *integration schema* contains additional information how the data and services of local servers are to be integrated into the grid. The integration schema does not duplicate the definitions from the contributory schemata. It holds only the items that cannot be included in the contributory schemata, e.g. the way to integrate pieces of a fragmented object the relationships among local servers that they are not aware of. The integration schema is used during the creation of views of the global virtual store.

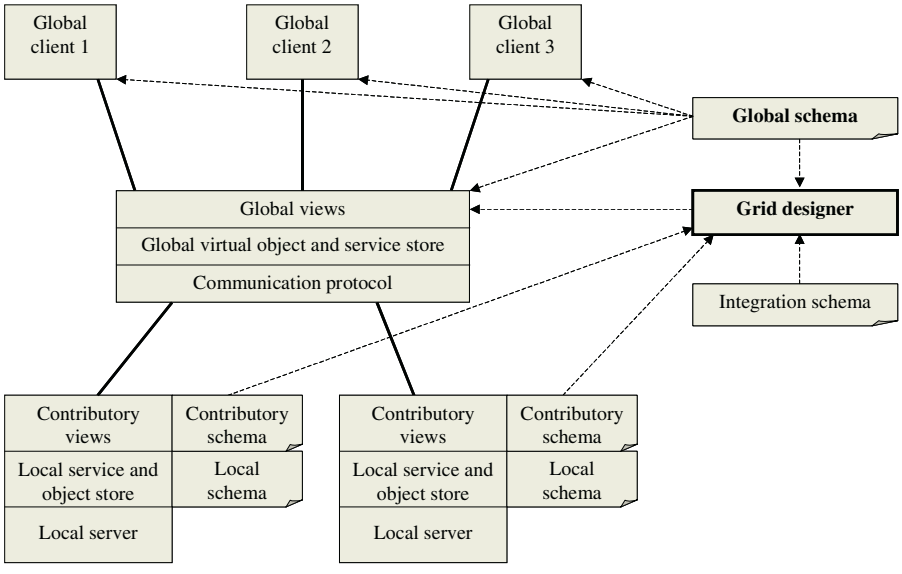


Fig. 2. Architecture of the GRID

The architecture of GRID is depicted in Fig. 2. In the figure solid lines represent run-time relationships i.e., queries, requests, and answers. Global users ask global view for resources and the global view request for some resources from local servers.

In the figure dashed arrows illustrate association that are used during development of the grid software. The programmers of global applications (global clients) use the global schema. The global views conform to the global schema. The grid designer uses the contributory schemata, the global schema and the integration schema in order to develop a global view.

4.2 Adaptation of the Approach to SBA

In the approach global views have to have access to information stored at local servers. Therefore, the global virtual store keeps the addresses of local servers in the form:

(*oid*, *server's name*, *server's address*)

In this triple *oid* is the internal identifier inside the global virtual store, *server's name* is a human-readable name (e.g. Cracow) and *the server's address* is the description of the physical location of the server (e.g. URI or URL). We call such a triple a *server link object*. A server link object looks like a link object. However, when someone navigates into it, it behaves like a complex object.

Let us assume that we have a server link object (i_C, *Cracow*, *IPCracow*). Conceptually when identifier i_C is processed (e.g. when query *Cracow.Book. title* is evaluated), ES is augmented with all root objects of the local server located at *IPCracow*. More formally the value of *nested*(i_C) is equal to the set of all the root objects stored by the server identified by i_C (in this case server *IPCracow*). This process should be

the subject of query optimization, to avoid sending to the global virtual store too many information. Conceptually all the root objects migrate onto the global virtual store's ES. Physically however local query engines can evaluate certain parts of queries (like *Book . title* in this case).

When object references fall onto the top of the global store's ES, they are no longer sufficient to identify objects. The local object identifiers are unique only inside local servers. Thus when a local object identifier i_O finds its way onto the global store's ES, it is implicitly converted to *global object identifier* i.e. pair (i_{LS}, i_O) where i_{LS} is the identifier of the server link object.

Therefore, the binders on the global store's ES can hold the identifiers of server link objects, the global object identifiers and possibly references of necessary local objects of the global virtual store.

5 Example Application

In this example we show that our approach can deal with replicas of data. We assume that we deal with multiple bookstores (respectively, located in Cracow, Warsaw, and Radom) and each of them sells books. We would like to create one virtual bookstore that offers books from all locations. Additionally, the local server in Cracow is a replica of the server in Warsaw.

In the integration schema we have the following information:

1. ISBN is a unique identifier of a book.
2. There are no duplicates in databases in Warsaw and Radom.
3. The local servers in Cracow and in Warsaw contain the same information.
4. Data should be retrieved from the server that has shorter access time.
5. Data in Cracow cannot be modified as they are replica of data in Warsaw.
6. After updating data in Warsaw data in Cracow are automatically updated.

The example global view is below:

```
create view myBookDef {
  virtual objects myBook {
      int timeTo Warsaw := 1000000; int timeToCracow := 1000000;
      if alive(Warsaw) then timeToWarsaw := checkAccessTime(Warsaw);
      if alive(Cracow) then timeToCracow := checkAccessTime(Cracow);
      if min(bag(timeToWarsaw, timeToCracow)) > 100 then {
            exception(AccessTimeToHigh);
            return ∅;
      }
      return  ((Radom . Book as b) ∪
            if timeToWarsaw < timeToCracow  then (Warsaw . Book as b)
                                            else  (Cracow . Book as b));
  }
  on_retrieve do {
      return b.(deref(ISBN) as ISBN, deref(title) as title,
            deref(author) as author, deref(price) as price)
  }
```

```
    on_delete do {
          if server(b) = Cracow then
              delete (Warsaw . Book where ISBN  = b.ISBN);
          else delete b;
    }
    on_insert newObj do {
          if server(b) = Cracow then
              insert newObj into ( Warsaw . Book where ISBN = b.ISBN);
          else insert newObj into b;
    }
    on_update newTitle do
          if server(b) = Cracow then
              (Warsaw . Book where ISBN = b.ISBN) . title := newTitle;
          else b . title := newTitle;
}}
```

Note, that in this example there are implicitly used some routines of the communication protocol i.e., navigation ("**.**"), deletion (keyword **delete**), and update (":="). All these operations are called as if they are performed locally, in fact they are performed on data from remote servers. In means that inside global virtual store these operations have to be implemented as elements of the communication protocol.

Apart from the core protocol routines (like navigation, insert, delete, and update), we used additional routines of communication protocol. Their names are underlined. Functions *alive* and *checkAccessTime* are used to determine which local servers (Cracow or Warsaw) to use when constructing the seeds. The routine *server* acts on a pair (i_{LS}, i_O) returned by query *b* and then delivers the name *server's name* from the triple (i_{LS}, *server's name*, *server's address*) representing the corresponding local server.

Clients can call the view in the following query:

(*myBook* **where** *title* ="Solaris") . (*author, price*)

that returns an author and a price of the book with the title "Solaris". This example shows that our mechanism supports service providers transparency – the user does not need to know the origin of the data.

6 Conclusion

We have presented a novel approach to implementation of federated databases based on updatable views. The approach fulfills some fundamental requirements for grid applications like transparency, interoperability and computational universality. The approach is based on a powerful query language SBQL which ensures high abstraction level. The advantage is decreasing development time of grid applications and simpler adaptation to changing requirements. The presented view mechanism is flexible and allows for describing any mapping between local databases and a federated database. Since our views can also map methods, we offer facilities as powerful as CORBA or Web Services. We have implemented SBQL and updatable views for XML repositories based on the DOM model. Our nearest plans assume implementation of our approach to the grid technology in the prototype ODRA, an object-

oriented database platform build from scratch by our team (ca. 20 researchers). Currently in ODRA we have finished implementation of SBQL and start to implement other functionalities (imperative constructs, procedures, views, distribution protocols, etc.) that are necessary to make our idea sufficiently ready for testing and prototype applications.

References

[Bell97] Z. Bellahsene: Extending a View Mechanism to Support Schema Evolution in Federated Database Systems. Proc. of DEXA 1997, 573-582

[Hal01] A. Y. Halevy: Answering queries using views: A survey. VLDB J. 10(4): 270-294 (2001)

[KLN+04] S. Kambhampati, E. Lambrecht, U. Nambiar, Z. Nie, G. Senthil: Optimizing Recursive Information Gathering Plans in EMERAC. J. Intell. Inf. Syst. 22(2): 119-153 (2004)

[KLS03] H.Kozankiewicz, J. Leszczyłowski, K. Subieta. *Updateable XML Views.* Proc. of ADBIS'03, Springer LNCS 2798, 2003, 385-399

[KW96] Chung T. Kwok, Daniel S. Weld: Planning to Gather Information. AAAI/IAAI, Vol. 1 1996: 32-39

[SKL95] K.Subieta, Y.Kambayashi, J.Leszczyłowski. Procedures in Object-Oriented Query Languages. Proc. of 21-st VLDB Conf., 1995, 182-193

[Subi00] K.Subieta. Mapping Heterogeneous Ontologies through Object Views. Proc. of EFIS, IOS Press, 1-10, 2000

Data Mining Tools: From Web to Grid Architectures

Davide Anguita, Arianna Poggi,
Fabio Rivieccio, and Anna Marina Scapolla

DIBE Dept. of Biophysical and Electronic Engineering,
University of Genoa, Via Opera Pia 11a, 16145 Genova, Italy
{anguita, apoggi, rivieccio, ams}@dibe.unige.it

Abstract. The paradigm of Grid computing is establishing as a novel, reliable and effective method to exploit a pool of hardware resources and make them available to the users. Data-mining benefits from the Grid as it often requires to run time consuming algorithms on large amounts of data which maybe reside on a different resource from the one having the proper data-mining algorithms. Also, in recent times, machine learning methods have been available to the purposes of knowledge discovery, which is a topic of interest for a large community of users. The present work is an account of the evolution of the ways in which a user can be provided with a data-mining service: from a web interface to a Grid service, the exploitation of a complex resource from a technical and a user-friendliness point of view is considered. More specifically, the goal is to show the interest/advantage of running data mining algorithm on the Grid. Such an environment can employ computational and storage resources in an efficient way, making it possible to open data mining services to Grid users and providing services to business contexts.

1 Introduction

The availability of large amounts of data has raised the interest in various tasks having the common goal of finding complex relationships between the supplied data. These tasks, often referred to as *data mining*, include (among others) classification, regression and clustering: the first two deal with the prediction of a value (in an ordered or unordered set) and the last is related to estimating a probability density. Machine Learning [19] provides methods and theory to perform these tasks in an inductive and data-based fashion, where a data-set is regarded as a collection of samples coming from an input/output statistical relation. This relation is *learnt* by means of algorithms working on data and the resulting Machine (e.g. a classifier or a regressor) can be used to forecast future output values on the basis of some input. The Support Vector Machine [8] is currently rated among the best performing algorithms and can be used to accomplish any of the three tasks above (and more). The recent inclusion of this algorithm in the data-mining suite of a well known database product [6] witnesses this method is fit for a large scale practical exploitation.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 620–629, 2005.

In this scenario, the paradigm of Grid computing can be seen as a natural and fruitful evolution of the software layer on which data-mining builds. Data mining algorithms need a large amount of storage and computational resources, and Grid computing is a way to meet these requirements. A Grid enabled environment should provide the core processing capabilities with secure, reliable and scalable high bandwidth access to the various distributed data sources and formats across various administrative domains [14]. When running computationally intensive processes in a dynamic Grid environment, a further advantage comes from having an accurate representation of the available resources and their current status. An example of the adoption of data–mining techniques in conjunction with a grid–enabled technology is given by the `NEOS` server [15].

The next section clarifies the main technical issues related to the SVM algorithm while section 3 describes in greater detail the two realizations of the SVM model selection and testing as an on-demand service. Paragraph 3.1 deals with the deployment of SVM as a web-accessible service, while paragraph 3.2 is focused on the SVM Grid service. The paper ends with a hint on possible future developments of the Grid service (Sec.4).

2 Data Mining Through Support Vector Machines

The Support Vector Machine was presented by Vapnik and Cortes [8] in the mid of the nineties. The method has proven an effective data mining tool since it was successfully adopted to solve problems in diverse fields. A predictive model is generated by induction on the sole basis of "examples", i.e. couples of input/ouput objects which underlie an unknown relation. The goal of the method is to learn a model for this relation.

This paradigm lies within the so-called *Statistical Learning Theory* [19], developed by Vapnik and Chervonenkis in the early seventies. The theory gives the method a sound background from various points of view:

- The procedure which identifies the final model is principled as it embeds a requirement of smoothness together with the obvious requisite to score a low number of errors.
- Effective criteria are available to select a model endowed with good generalization capability. Novel methods which are theoretically well founded drive the choice of a performing model [4, 2].
- Performance bounds holding in probability exist to estimate the error rate scored on-line by the generated model [3].

The Support Vector Method provides algorithms to tackle very different sorts of problem: classification, regression and clustering. Each of these problems is casted into a constrained quadratic programming (CQP) problem offering many advantages, such as the existence of a unique solution and the availability of many algorithms to solve it in a fast and accurate way.

For the reader's convenience the SVM algorithm for classification is here detailed, anyhow the same way of reasoning can be likewise applied in the other

Table 1. Valid kernel functions

Kernel Type	Kernel expression
Linear	$k(x_i, x_j) = x_i \cdot x_j$
Gaussian	$k(x_i, x_j) = e^{-\gamma \lvert x_i - x_j \rvert^2}$
Polynomial	$k(x_i, x_j) = (\frac{(x_i \cdot x_j)}{ni} + 1)^p$

two data-mining tasks. Two issues are considered when seeking for a good Support Vector Classifier: a low number of training errors and a smooth classifier. These two requirements are embedded in an optimization problem, called *primal problem*. The solution for this problem, if $\mathbf{x} \in \Re^{ni}$, is of the form

$$f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b \tag{1}$$

and $|\mathbf{w}|^\mathbf{2}$ is a smoothness measure: the lower the norm, the better the generalization capability. Actually, the margin between the two classes scales as the inverse of $|\mathbf{w}|^\mathbf{2}$, hence searching for a smooth function means searching for a large margin classifier. Extension to non-linear classification is possible through the so-called *kernel trick* [1]: the basic idea is to replace each dot product appearing in the problem formulation with a suitable *kernel function* thus enabling a non-linear separation of the data. The theoretical justification of this peculiar procedure is to map the data-points into a high dimensionality Hilbert space, where the chance to find a good linear separation is greatly augmented. The linear separation found in this mapped space (called *feature space*) is then mapped back into the input space and can be described by only means of dot products between mapped points: the dot product in the features space can be expressed by a kernel function in the input space, thus allowing to describe the linear separation in the feature space as a non-linear separation in the input space [7]. The (implicit) seek for a suitable hyperplane in the feature space and thus the identification of a non-linear classifier is made possible by the solution of a problem originating from the primal, called the *dual problem* which contains only dot products between input points (thus making possible the use of the kernel trick). The dual problem is a CQP problem and its solution is given by a set of lagrange multipliers related to the optimization problem (α_i) which minimize the dual cost function and define the classifier in the following way:

$$f(\mathbf{x}) = \sum_{i=1}^{np} \alpha_i y_i k(x_i, x) + b \tag{2}$$

where b is a bias, the value of which can also be assessed from the alphas, and $k(x_i, x)$ is the kernel function. Suitable kernels are:

It turns out that often the solution is sparse (some of the α_i are zero), then the solution can be described (or *supported*) by a number of points less than np, i.e. those points corresponding to the non-zero alphas; for this reason these points are also called *support vectors*. A by product of the whole SVM algorithm is then a selection of the input points which are most relevant to the sake of

classification. The final form of the solution is a weighted sum involving the kernel function (see eq.2), where the CQP problem identifies the weights (i.e. the alphas) of the sum. Some other parameters are clearly not identified by the optimization procedures but must be set a priori in order to instruct the CQP. These parameters are sometimes called for this reason *hyperparameters* and they are:

- The values shaping the kernel function (e.g. for RBFs the value of the width, for polynomials the degree, etc.).
- The upper bound on the alpha value (as they should not grow to an infinite value); this also results in setting the overall error relevance.
- A parameter modeling the noise amount (only for regression problems).

Hyperparameters are tightly connected to the final system performance. Statistical Learning Theory features some probabilistic bounds which estimate the on-line error rate related to a certain model; as a by product, one can choose the model (i.e. a certain set of hyperparameters) on the basis of the forecasted performance. Many methods are available to this end; the *resampling* methods (e.g. the *bootstrap* [11]) are among the most used: they build on the concept of splitting the available data into subsets that are used for training and testing various classifiers. The on-line performance is then devised on the basis of the average test error. Despite the problem of how to select a model is quite thorny, practical methods often grant a fair performance: for an account of the vast amount of real-world problems solved through the use of SVM see [13].

3 Deploying a SVM-Based Classifier as an On-demand Service

From the above description it is clear that some efforts must be made to shrink the gap between unskilled users and data mining algorithms. The two following examples show how the Support Vector Algorithms can be offered to users in a friendly way: we first present the Internet-based Smart Adaptive Algorithm Computational (ISAAC) server, and then the implementation of SVM as a Grid service.

The goals shared by these two implementations are:

- No specific knowledge of what the hyperparameters are and of their values is required: the system seeks for the best values in a transparent way.
- The user is allowed to upload his own data without bearing the load of computation required by the algorithm: the mining engine resides in fact on a remote server.
- The architectural layout of the whole system should be easy to upgrade in order to widen the offered solution portfolio with new mining algorithms and facilities.

The main intent that motivates the present work is to set-up an on-demand data-mining service exploiting remote resources and freeing users from the necessity of having expensive hardware and/or a solid know-how in the field of

Machine Learning. Indeed, the remote elaboration is here carried on in the most user-transparent possible way, only requiring the user to perform some basic actions (like identification and data upload) in order to build a proper classifier.

3.1 The ISAAC Server

The ISAAC server hosts a sequence of web pages which allow a user to interactively launch a model selection procedure for finding a Support Vector Classifier based on the data provided by the user himself. The server builds on a cluster of PCs connected through a switch, thus avoiding possible bottlenecks. The user can connect to the system at the following URL:

`http://www.smartlab.dibe.unige.it/`

The system features 8 Intel P4 @ 2.4/2.8GHz nodes with 1GB RAM and an overall storage space of about a TeraByte. Each node of the cluster runs Linux Red Hat 9 and the hosting is granted by the `Apache` web server. Users are able to access the system (without having to register in the system and supply a password) and upload data through an upload form. The web connection offers the standard security provided by the `https` protocol. In this case the user navigates in different forms which allow him/her to upload the data, launch an elaboration including model selection with historical data and test the built model on new data. The user is finally sent the communication of the estimated output values via e-mail. A software infrastructure made of `bash` procedures called via `php` provides the necessary substrate for the system to function properly. No user registration other than email address storing has been provided here. In compliance with the desired features, the ISAAC system can be updated with new mining algorithms without affecting the general process architecture, as all the requests are forwarded to a unique Requests Queue Manager (RQM). This module decides when the overall load is low enough to launch another process: it gains information about each node load state through monitoring facilities and passes to the Process Launching Module (PLM) the reference to the job request. The PLM launches the jobs and is responsible to instruct the Results Delivering System (RDS) about which user has to be contacted when a certain job ends. The general architecture is depicted in Fig.1.

All the procedures running in the ISAAC system are specific and cannot be ported without a significant intervention to fit the code to novel hardware resources. Also, the addition of new nodes to the cluster is tightly bounded to the OS, which should run some Linux distribution in order to avoid a time consuming tuning phase. In other words, a scaling of the whole system to a wider cluster of resources could not be easily performed. As already pointed out, the security of the system (which is critical when uploading sensible data) is demanded to the standard `https` protocol. All the problems presented are commonplace in a cluster of workstations: in particular, the security issues are wholly demanded to the server, which should filter malicious actions coming from the outer world; security issues among the cluster nodes are clearly not

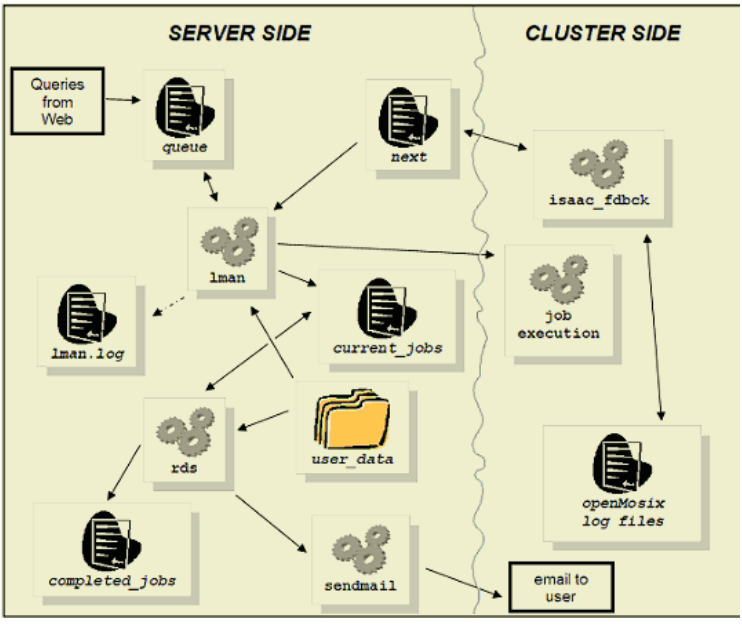


Fig. 1. The ISAAC process architecture

considered. The paradigm of Grid computing addresses these problems (among others) offering a more reliable and effective solution.

3.2 SVM as a Grid Component

Grid computing is establishing as the state-of-the-art technology to connect remote and heterogeneous resources and make them available to the users' community. The scenario opened by Grid computing features a dynamic optimization of the computational load across the whole pool of resources, also allowing the delivering of payback services in a business framework. A Grid infrastructure at DIBE (Department of Biophysical and Electronic Engineering) is being built up [10] as a embryonal testing environment for Grid distributed technology with a single administrative domain. This specific Grid is focused on data-mining problems, and the tool considered for tackling such problems is the SVM algorithm. The Grid structure also allows power users to add new mining tools so to set up a modular mining framework without having to complement every new method with some ad-hoc code. The environment is open to evolution incorporating other nodes (e.g. university departments or business organizations) and it is able to become an effective Grid-based system, geographically distributed and comprehensive of different organizations each one with its distinctive network and security policies. The hardware architecture is composed by some nodes running different Linux distributions and the Globus Toolkit v.3.2 middleware: the *de facto* standard to build a Grid infrastructure. One of these nodes is consti-

tuted by the above described Isaac System. More precisely, only the web server is a Grid node and the cluster remains in a private sub-network. The idea is to extend and improve the Isaac system, that has shown security and scalability limits, providing an on-demand data-mining service, a secure, reliable, scalable and portable service for the Grid users. In fact, the requirements for a data-mining algorithm are a secure transfer of data between the client application (user interface) and the Grid node, as well as a large amount of computational resources for a quick response to the user. The requests of data-mining elaborations are met through Grid Service techniques: a distributed computing technology that allows creating client/server applications and obtaining interoperability between different operating systems, programming languages and object models. Grid Services are an extension of Web Services technology so they inherit all advantages of that technology plus some features. They are platform-independent and language-independent, since they use the standard XML language and the Internet standard HTTP (HyperText Transfer Protocol). Distributed technologies already existing like CORBA, RMI, EJB, result in highly coupled distributed systems, where the client and the server are very dependent on each other; Web Services are instead oriented towards loosely coupled systems and meet the demands of an Internet-wide application, just like Grid-oriented applications [12]. Until now Grid Services can be regarded as a "patch" over the Web Services (WS) specification, but from the beginning of this year, a new WS specification

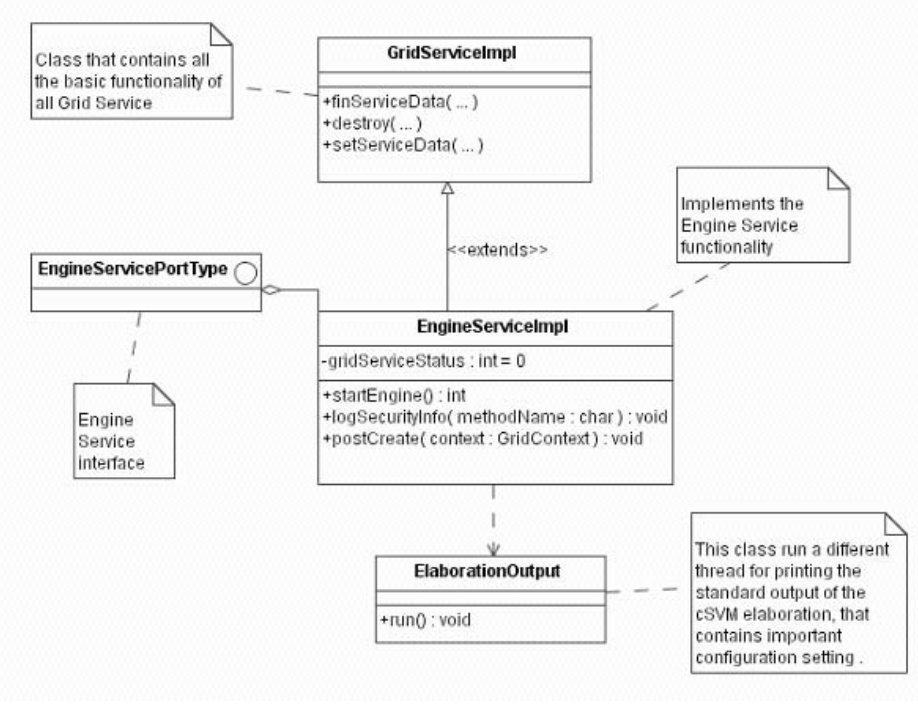


Fig. 2. Engine Service Class Diagram

is coming up: the WSRF, Web Services Resource Framework [20]. This framework defines a family of specifications for accessing stateful resources using Web services and includes the WS-ResourceProperties, WS-ResourceLifetime, WS-BaseFaults, and WS-ServiceGroup specifications. The motivation for this is that while Web service implementations typically do not maintain the state information during their interactions, their interfaces have the need to frequently allow for the manipulation of the state, that is, data values that persist across and evolve as a result of Web service interactions [18]. The choice of developing a Grid Service is justified by the great portability, scalability and security of this kind of applications. In this secure environment, only accessible to authorized logins (i.e. certificated by a custom DIBE Certification Authority), users requiring data mining services have the peacefulness that their sensible data are transferred and elaborated over a secure environment. The service is called Engine Service (like the name of its PortType), and its class EngineServiceImpl extends Grid ServiceImpl, the base class that provides the basic functionality for a Grid Service. The only exposed method is startEngine() that starts the SVM elaboration on the Grid. The other two methods are not exposed in the Grid Service but they implement the service security (see Fig.2). A client application is provided to give a user-friendly interface to interact with the service. From this one, it is possible to transfer the input data files and start the elaboration, involving the identification and building of a proper model for a support vector classifier. The user can upload two files: one is used for training and is made of couples of input/output values, the other is only composed of input values (called *validation* data), the task of the classifier being an estimation of the correct labels. If the user does not set any of the hyperparameters, the algorithm performs the model selection procedure and seeks for the set of hyperparameters which minimizes the estimated generalization error. The results are displayed in two text areas: one for the model, which supplies detailed information about the learning phase, and one for the validation, which tells how fairly the model is performing on the validation data. The mechanism of file transfer is implemented using GridFTP: a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. The GridFTP protocol is based on FTP, the highly-popular Internet file transfer protocol [16](see Fig.3).

4 Final Considerations

The system presented above is oriented towards on-demand computing. A future refinement in the sense of a dynamic and effective distribution of the computational load will improve the overall quality of service offered to users. In a scenario in which one tries to exploit both the Grid capabilities and the cluster computational power, this first layer could be exclusively managed by the Grid. An evolution is planned to grow the complexity of the software architecture to enable a faster model selection procedure: once the data is uploaded by the user, the estimation of the generalization ability of the various candidate models could be spread across the available resources; this second layer could

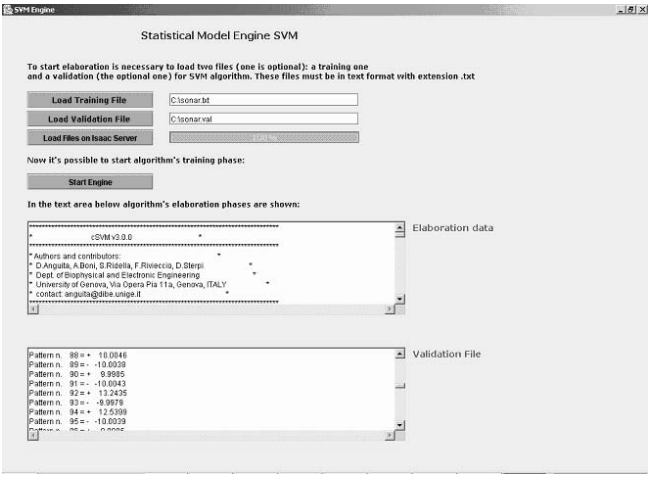


Fig. 3. Engine Service Client Interface Appearance

be managed both by the Grid level and the cluster one. Each evaluation can be performed in a parallel fashion thus speeding up the identification of the estimated best model. Some methods are also available in literature to speed the CQP optimization based on previously available solutions related to a different set of hyperparameters (e.g. *alpha seeding* procedures [5]).

Besides these two layers into which the service can be allotted, a third and deeper level of parallelization can be foreseen. Recently some algorithms have been made available ([21]) which allow to parallelize each CQP problem, thus opening to new scenarios in which a single classification request will be distributed into several tasks on the basis of a two-folded parallelization: the classification task is at first split into the parallel testing of several potential models and secondly the evaluation of each of these models is shared among a number of nodes which contribute to solving the same CQP problem. This third layer could be exclusively performed by the cluster, thus enabling better performance.

The challenges lying in this envision are the proper allocation of the tasks coming from different levels of optimization and the handling of concurrent requests coming from different clients, in order to lower the response time perceived by the single user.

References

1. M. A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.
2. D. Anguita, S. Ridella, F. Rivieccio, R. Zunino, Hyperparameter design criteria for support vector classifiers, *Neurocomputing*, Vol. 55, N. 1-2, pp. 109-134, 2003.

3. D. Anguita, A. Boni,D. Sterpi, S. Ridella, F. Rivieccio, *Theoretical and Practical Model Selection Methods for Support Vector Classifiers*, in "Support Vector Machines: Theory and Applications", by L. Wang (Ed.), Springer, in press.
4. P. L. Bartlett, S. Boucheron, and G. Lugosi. Model selection and error estimation. *Mach. Learn.*, 48(1-3):85–113, 2002.
5. D. DeCoste, K. Wagstaff, Alpha seeding for support vector machines, *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pp.345–359, 2000.
6. Oracle Corp. Discover patterns, make predictions, develop advanced BI A pplications, January 2004.
7. M. G. Genton, Classes of Kernels for Machine Learning: A Statistics Perspective, *Journal of Machine Learning Research*, vol. 2, pp. 299-312, 2001.
8. C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
9. D. Anguita, A. Boni, and S. Ridella. Evaluating the generalization ability of support vector machines through the bootstrap. *Neural Processing Letters*, 11(1), 2000.
10. D. Anguita, A. Poggi, and A.M. Scapolla. Smart adaptive algorithm on the grid. *11th Plenary HP-OUVA Conference, June 2004, Paris*, page 2, 2004.
11. B. Efron and R. Tibshirani. *An Introduction to the Bootstrap.* Chapman and Hall, London, 1993.
12. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.
13. I. Guyon. Online svm application list, available on web at: http://www.clopinet.com/isabelle/projects/svm/applist.html.
14. BeSC-Belfast e-Science Centre Online available on web at: http://www.qub.ac.uk/escience/projects/geddm/.
15. J. Czyzyk, M. P. Mesnier, J. J. Mor, "The NEOS Server", *IEEE Computational Science and Engineering*, Vol.5, N.3, pp. 68-75, 1998.
16. Globus-Project, "GridFTP, Universal Data Transfer for the Grid", *White Paper*, Sept. 2000.
17. T. Poggio and S. Smale, The mathematics of learning: Dealing with data, *Amer. Math. Soc. Notice*, Vol.50(5),pp. 537–544, 2003.
18. IBM-Report. Web services resource framework, March 2004.
19. V. Vapnik. *Statistical Learning Theory.* Wiley, 1998.
20. I. Foster, J. Frey, S. Graham, S. Tuecke, K.Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T.Storey, W.Vambenepe, S.Weerawarana, Modeling Stateful Resources with Web Services, whitepaper available at: http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf
21. G. Zanghirati, L. Zanni, "A Parallel Solver for Large Quadratic Programs in Training Support Vector Machines", *Parallel Computing*, 29 (2003), 535-551.

Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications*

Cosimo Anglano and Massimo Canonico

Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria, Italy
{cosimo.anglano, massimo.canonico}@unipmn.it

Abstract. In this paper we propose a fault-tolerant scheduler for Bag-of-Tasks Grid applications, called *WorkQueue with Replication Fault Tolerant* (WQR-FT), obtained by adding checkpointing and replication to the *WorkQueue with Replication* (WQR) scheduling algorithm. By using discrete-event simulation, we show that WQR-FT not only ensures the successful completion of all the tasks in a bag, but also achieves performance better than WQR and other fault-tolerant schedulers obtained by coupling WQR with replication only, or with checkpointing only.

1 Introduction

Grid Computing technology provides resource sharing and resource virtualization to end-users, allowing for computational resources to be accessed as a utility. By dynamically coupling computing, networking, storage, and software resources, Grid technology enables the construction of virtual computing platforms capable of delivering unprecedented levels of performance. However, in order to take advantage of Grid environments, suitable application-specific scheduling strategies, able to select, for a given application, the set of resources that maximize its performance, must be devised [2]. The inherent wide distribution, heterogeneity, and dynamism of Grid environments makes them better suited to the execution of loosely-coupled parallel applications, such as *Bag-of-Tasks* [11] (BoT) applications, rather than of tightly-coupled ones. Bag-of-Tasks applications (parallel applications whose tasks are completely independent from one another) are particularly able to exploit the computing power provided by Grids [6] and, despite their simplicity, are used in a variety of domains, such as parameter sweep, simulations, fractal calculations, computational biology, and computer imaging. Therefore, scheduling algorithms tailored to this class of applications have recently received the attention of the Grid community [3, 4, 6]. Although these algorithms enable BoT applications to achieve very good performance, they suffer from a common drawback, namely their reliance on the assumption that the resources in a Grid are perfectly reliable, i.e. that they will

* This work has been supported by the Italian MIUR under the project *Societá dell'Informazione, Sottoprogetto 3 - Grid Computing: Tecnologie abilitanti ed applicazioni per eScience*, L. 449/97, anno 1999.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 630–639, 2005.

never fail or become unavailable during the execution of a task. Unfortunately, in Grid environments faults occur with a frequency significantly higher than in traditional distributed systems, so this assumption is overly unrealistic. A Grid may indeed potentially encompass thousands of resources, services, and applications that need to interact in order for each of them to carry out its task. The extreme heterogeneity of these elements gives rise to many failure possibilities, including not only independent failures of each resource, but also those resulting from interactions among them. Moreover, resources may be disconnected from a Grid because of machine hardware and/or software failures or reboots, network misbehaviors, or process suspension/abortion in remote machines to prioritize local computations. Finally, configuration problems or middleware bugs may easily make an application fail even if the resources or services it uses remain available [9].

In order to hide the occurrence of faults, or the sudden unavailability of resources, fault-tolerance mechanisms (e.g., *replication* or *checkpointing-and-restart*) are usually employed. Although scheduling and fault tolerance have been traditionally considered independently from each other, there is a strong correlation between them. As a matter of fact, each time a fault-tolerance action must be performed, i.e. a replica must be created or a checkpointed job must be restarted, a scheduling decision must be taken in order to decide where these jobs must be run, and when their execution must start. A scheduling decision taken by considering only the needs of the faulty task may thus strongly adversely impact non-faulty jobs, and vice versa. Therefore, scheduling and fault tolerance should be jointly addressed in order to simultaneously achieve fault tolerance and satisfactory performance. *Fault-tolerant schedulers* [1, 12, 15] attempt to do so by integrating scheduling and fault management, in order to properly schedule both faulty and non-faulty tasks. However, to the best of our knowledge, no fault-tolerant scheduler for BoT applications has been proposed in the literature. This paper aims at filling this gap by proposing a novel fault-tolerant scheduler for BoT applications, that we call *WorkQueue with Replication/Fault-Tolerant* (WQR-FT). WQR-FT extends the *WorkQueue with Replication* (WQR) [6] algorithm by adding to it both task checkpointing and automatic restart of failed tasks. As shown by our results, the simultaneous adoption of these mechanisms, besides providing fault tolerance, allows WQR-FT to achieve performance better than WQR, WQR using automatic restart only, and WQR using checkpointing-and-restart only. The rest of the paper is organized as follows. In Section 2 we present the WRQ-FT scheduling algorithm, while in Section 3 we presents the performance results we obtained in our experiments. Finally, Section 4 concludes the paper and outlines future research work.

2 WQR-FT: A Fault-Tolerant Scheduler for BoT Applications

Scheduling applications on a Grid is a non trivial task, even for simple applications like those belonging to the BoT paradigm [6]. As a matter of fact, the

set of Grid resources may greatly vary over time (because of resource additions and/or removals), the performance a resource delivers may vary from an application to another (because of resource heterogeneity), and may fluctuate over time (because of resource contention caused by applications competing for the same resource). Achieving good performance in these situations usually requires the availability of good information about both the resources and the tasks, so that a proper scheduling plan can be devised. Unfortunately, the wide distribution of Grid resources makes obtaining this information very difficult, if not impossible, in many cases. Thus, the so-called *knowledge-free* schedulers, that do not base their decisions on information concerning the status of resources or the characteristics of applications, are particularly interesting.

WorkQueue with Replication is a knowledge-free scheduling algorithm that adds task replication to the classical *WorkQueue* scheduler. Our scheduler, WQR-FT, adds both automatic restart and checkpointing to WQR, and properly coordinates the scheduling of faulty and non-faulty tasks in order to simultaneously achieve fault-tolerance and good application performance.

2.1 The Standard WQR Scheduler

In the classical WorkQueue (WQ) scheduling algorithm, tasks in a bag are chosen in an arbitrary order and are sent to the processors as soon as they become available. WQR adds task replication to WQ in order to cope with task and host heterogeneity, as well as with dynamic variations of the available resource capacity due to the competing load caused by other Grid users. WQR works very similarly to WQ, in the sense that tasks are scheduled the same way. However, after the last task has been scheduled, WQR assigns replicas of already-running tasks to the processors that become free (in contrast, WQ leaves them idle). Tasks are replicated until a predefined *replication threshold* is reached. When a tasks replica terminates its execution, its other replicas are canceled. By replicating a task on several resources, WQR increases the probability of running one of the instances on a faster machine, thereby reducing task completion time. As shown in [6], WQR performance are equivalent to solutions that require full knowledge about the environment, at the expenses of consuming more CPU cycles.

2.2 The WorkQueue with Replication – Fault Tolerant Scheduler

In its original formulation, WQR does not do anything when a task fails. Consequently, it may happen that one or more tasks in a bag will not successfully complete their execution. In order to obtain fault tolerance, we add *automatic restart*, with the purpose of keeping the number of running replicas of each task above a predefined replication threshold R. In particular, when a replica of a task t dies and the number of running replicas of t falls below R, WQR-FT creates another replica of t that is scheduled as soon as a processor becomes available, but only if all the other tasks have at least one replica running. Automatic restart ensures that all the tasks in a bag are successfully completed even in face of resource failures. However, each time a new instance must be

started to replace a failed one, its computation must start from the beginning, thus wasting the work already done by the failed instance. In order to overcome this problem, WQR-FT uses *checkpointing*, that is the state of the computation of each running replica is periodically saved with a frequency set as indicated in [13] (we postulate the existence of a reliable checkpoint server where checkpoints are stored). In this way, the execution of a new instance of a failed task may start from the latest available checkpoint. In this paper we assume that an application may be restarted from its latest checkpoint only on a machine having the same operating system of that on which the checkpoint was taken. If such a machine cannot be found when a new replica is created (to replace a faulty one), its execution will start from the beginning.

3 Performance Analysis

In order to assess the performance of WQR-FT, we compared it with both plain WQR, WQR using automatic restart only (henceforth referred to as *WQR-R*), and WQR using checkpointing only for various values of the replication threshold and checkpoint overhead, as well as for a variety of different workloads. In order to perform this comparison, we developed a discrete-event simulator, based on the CSIM [10] libraries, that has been used to perform a large number of experiments for a variety of operational scenarios. Our comparison is based on the following two metrics:

- Average *task response time*, defined as the time elapsing between the submission of the bag to which the task belong and the successful completion of its *first instance*;
- Average *BoT completion time*, defined as the time elapsing between the submission of the bag and the termination (either successful or not) of all its tasks.

3.1 Simulation Scenarios

Because the space covering the relevant parameters of our study is too large to explore exhaustively, we fix a number of system characteristics. The configuration of the Grid used in all the experiments has been obtained by means of GridG [5], that was used to generate a configuration comprising 100 clusters, each one comprising a random number of machines uniformly distributed between 1 and 16 (for a total number $N = 185$ of machines). The tasks submitted to each cluster are scheduled according to the FIFO policy. Individual machines do not run local jobs and do not support time-sharing. Each machine is characterized by its computing power P, an integer parameter whose value is directly proportional to its speed (i.e., a machine with $P = 2$ is twice faster than a machine with $P = 1$), that we assume to be uniformly distributed between 1 and 20. Moreover, each machine is associated with its operating system type, that is chosen with equal probability in the set $\{Solaris, Linux, FreeBSD\}$. A machine may

fail to complete the execution of its running task either because of a reboot (in 80% of the cases) or a crash (in 20% of the cases). The *Fault Time* (i.e., the time elapsing between two consecutive failures) is assumed to be a random variable with a Weibull distribution (in according with the results reported in [7]), while the *Repair Time* (i.e., the time elapsing from the fault to when the machine is operational again) is assumed to be uniformly distributed between 120 and 600 seconds (for a reboot) [8], or exponentially distributed with mean 2 days (for a hardware crash) [14]. The parameters of the Weibull distribution characterizing Fault Time were set according to the following procedure. We considered various scenarios in which the availability α of machines, defined as $\alpha = \frac{MTBF}{MTBF+MTTR}$ (where $MTBF$ and $MTTR$ denote the mean of the Fault Time and of the Repair Time, respectively), was set to $90\%, 50\%, 25\%$ and 10%, respectively. For a particular scenario, given the corresponding value of α and of the $MTTR$ (that in all our experiments was set to 34848 sec., according to the values reported before for the distributions of the Repair Time), we computed the $MTBF$ from the definition of α and, from this value, we obtained the parameters of the Weibull distribution.

3.2 The Workloads

In order to make an exhaustive comparison, we considered a rather large set of workloads, obtained by varying some parameters of a *base workload*. The base workload consists in a sequence of Bag of Tasks, each one comprising $RR \cdot N$ tasks, where N is the number of machines in the Grid ($N = 185$), and RR is a numerical coefficient used to change the size of each bag. The duration of each task is assumed to be a random variable uniformly distributed in the interval $[RT{*}0.5{*}BaseTime, RT{*}1.5{*}BaseTime]$ seconds, where $BaseTime$ and RT are workload parameters (note that the duration of tasks is referred to a machine with computing power $P = 1$). By suitably setting RR, RT, and $BaseTime$, very different workloads may be generated. In particular, in our study RR took values in the set $\{0.25, 0.5, 1, 1.25, 1.5\}$, RT in the set $\{0.5, 1, 1.5, 2, 3, 5\}$, and $BaseTime$ in the set $\{35000, 100000\}$, thus generating 70 different workloads.

3.3 Results

Let us now describe the results we obtained in our experiments. We first show that simple replication, as the one used by WQR, cannot guarantee the completion of all the tasks in a bag, except in scenarios characterized by high machine availability and by the use of a relatively high number of task replicas. Then, we will show that WQR-R outperforms WQR for the same number of replicas, and that WQR-FT outperforms both strategies for all the simulation scenarios we considered. In all our experiments, we computed 98% confidence intervals with a relative error of 2.5% or less for both the average task response time and the average BoT completion time.

Is Replication Sufficient? In order to verify whether the addition of further fault tolerance mechanisms to WQR is motivated by a real need, we per-

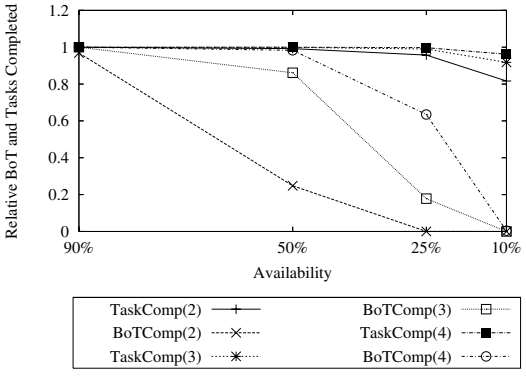


Fig. 1. Fraction of tasks and of BoTs successfully completed by WQR

formed a set of experiments in which we progressively decreased the availability of machines, and computed the fraction of tasks ($TaskComp(x)$) and of BoTs ($BoTComp(x)$) completed for various values of the replication threshold x. Our results, shown in Fig. 1 for a workload in which $RR = RT = 1$ (for the other workloads the results were not significantly different), clearly indicate that 100% of tasks (and, consequently, bag of tasks) is completed only with 4 replicas and for availability values greater than 90%. In all the other cases, the fraction of both completed tasks and bags decreases for decreasing availability values. This is due to the fact that the lower the availability, the higher the frequency of faults of each resource, and the probability of having all the instances of a task fail. Therefore, WQR without additional fault-tolerance mechanisms cannot ensure the successful completion of all the tasks, except in cases when machine availability is quite high, and a relatively large number of replicas is used.

WQR-R vs. WQR: The Benefits of Automatic Restart. Let us now compare the performance of WQR with those attained by WQR-R. Let us start with the results obtained, for various replication threshold values, for a workload in which $RR = 1$ (that is, each bag contains as many tasks as the number of Grid resources) and $RT = 1$. Fig. 2(a) and (b), where the number between parentheses indicate the value of the replication threshold, show respectively the average task response time and BoT completion time obtained by WQR and WQR-R for decreasing values of availability α and for $BaseTime = 35000$ sec. (the results for $BaseTime = 100000$ sec. are not significantly different, so are not reported here because of space constraints). As can be observed from the above figure, for availability values from 90% down to 25%, the average task response time (Fig. 2(a)) is practically not affected neither by the presence of automatic resubmission nor by the number of replicas, while a small difference in favor of WQR appears for $\alpha = 10\%$. This means that the presence of additional replicas created to replace faulty ones does not significantly increase the time

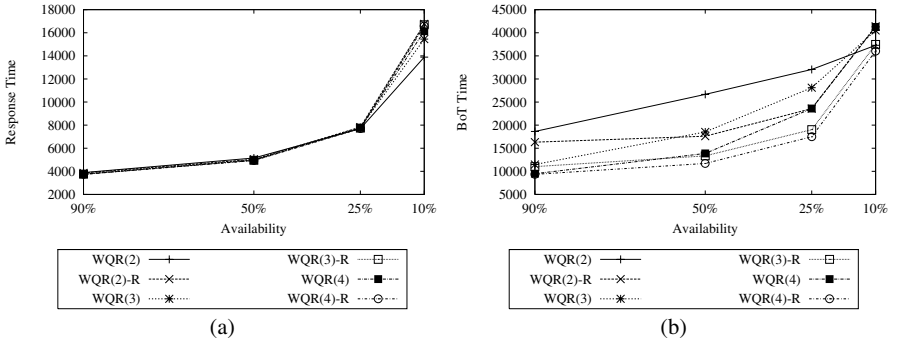


Fig. 2. (a) Average task response time, and (b) Average BoT completion time

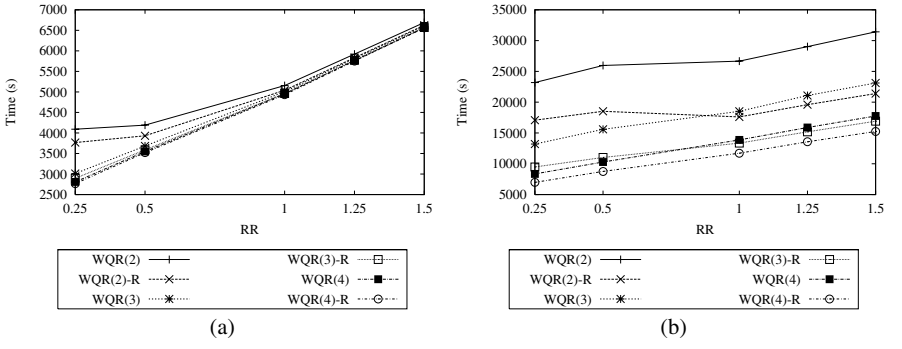


Fig. 3. (a) Average task response time, and (b) Average BoT completion time, $\alpha = 0.5$, RT=1

each tasks spends in the queue waiting to be scheduled. However, WQR-R results in a better average BoT completion time (Fig. 2(b)) than WQR for the same number of replicas. This is due to the fact that, when a replica fails, the one that replaces it may be scheduled on a faster machine, thus reducing the task execution time and, hence, the time taken to complete a bag. In order to validate this hypothesis, we performed a set of experiments in which we set $\alpha = 50\%$ and $RT = 1$, while we increased the size of the BoTs from 50% to 150% of the number of Grid resources (by varying RR accordingly). Our results show comparable task response times (Fig. 3(a)), but significants gains in terms of BoT completion time (Fig. 3(b)), especially for lower values of RR. This is due to the fact that the lower RR, the higher the number of machines on which a restarted replica may be executed, and consequently the higher the probability of choosing, for this replica, a faster machine. Therefore, our intuition is confirmed by these results.

WQR-FT vs. WQR-R: The Benefits of Checkpointing. As discussed in Sec. 2.2, the rationale behind the development of WQR-FT is to exploit the

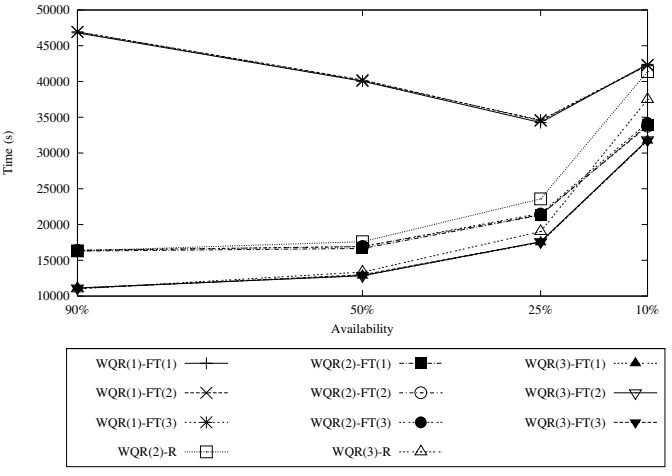


Fig. 4. Average BoT time

work already done by failed task replicas, so that better application performance may be obtained. In order to verify whether the introduction of checkpointing is beneficial or not, we performed a set of experiments in which we compared the performance of WQR-FT and WQR-R for various values of the checkpoint overhead (i.e., the time that each task is blocked to carry out a checkpoint operation) and of the replication threshold. More specifically, we run three sets of experiments in which the checkpoint overhead was assumed to be uniformly distributed between 0.1 and 0.3 sec. (*Set 1*), 6 and 10 sec. (*Set 2*), and 11 and 20 sec. (*Set 3*), respectively. In Fig. 4, where the first number between parentheses denotes the replication threshold, and the second one the experiment set (i.e., 1, 2, and 3 correspond to *Set 1, Set 2*, and *Set 3*, respectively), we report the results corresponding to the workload obtained by setting $RR = 1$, $RT = 1$, and $BaseTime = 35000$ sec., for different values of the availability α. As can be seen from this figure, the curves form three rather distinct groups, each one corresponding to a different replication threshold. The lower, intermediate, and upper group corresponds to scheduling strategies using 3 instances, 2 instances, and 1 instance for each task, respectively. As can be observed from these curves, the larger the number of replicas, the better the performance (the performance for larger values of the replication threshold show the same behavior, and have not included in the graph for the sake of readability). Within a given group is very hard to distinguish the curves corresponding to the various scheduling algorithms for availability values from 90% down to 50%, and this means that there is practically no difference among them. However, for lower availability values, checkpoint proves to be beneficial in terms of performance, and obviously the smaller the checkpointing overhead, the larger the performance gains. This is due to the fact that the higher the fault frequency, the higher the probability

that a task has already performed a large fraction of his work before failing. In these cases, the use of checkpointing is crucial to avoid wasting a large amount of already performed work. This effect is even more evident for workloads where $BaseTime = 100000$ s. (not shown here because of space constraints), since in these cases the amount of work wasted if checkpointing is not used is even larger. It is worth to point out that checkpointing alone is not sufficient to ensure good performance, as indicated by the upper set of curves shown in Fig. 4. These curves indeed show that simple checkpointing (that corresponds to WQR(1)-FT, i.e. to the use of a single task instance) attains the worst performance among all the considered alternatives. The graph concerning instead the task response time (not shown here because of space constraints), shows no difference – for a given replication threshold – among all the strategies, meaning that checkpointing does not enlarge the time spent by each tasks waiting to be scheduled. Therefore, we can conclude that, being the availability of machines usually unknown, it is better to use checkpointing rather than not, as this may result in much better performance when machine availability is low.

The results obtained for the other workloads and scenarios, not reported in this paper because of the limited amount of available space, confirm the above conclusions. In particular, we observed that the larger the execution time of tasks (i.e. the larger the values of RT and $BaseTime$), the larger the performance gains due to checkpointing. This is due to the fact that the larger the task execution time, the larger the performance loss occurring when a task replica has to restart its execution from the beginning. Moreover, the larger the value of RR, that is the number of tasks in a bag, the better the performance of checkpointing. This can be explained by the fact that, the larger RR, the lower the probability for WQR-R of finding soon an available machine faster than the one that failed. Conversely, in these situations the possibility of exploiting the work already done given by the use of checkpointing may significantly enhance performance.

4 Conclusions and Future Work

In this paper we have presented WQR-FT, a fault-tolerant scheduling algorithm for Bag-of-Tasks Grid applications based on the WQR algorithm. By simultaneously using replication and checkpointing, WQR-FT is able not only to guarantee the completion of all the tasks in a bag, but also to achieve performance better than alternative scheduling strategies. As a matter of fact, being WQR able to attain performance higher than other (non fault-tolerant) alternative strategies [6], and being WQR-FT to achieve performance better than WQR, we can conclude that WQR-FT outperforms these strategies when resource failures and/or unavailabilities are taken into account.

As future work, we plan to study the behavior of WQR-FT in more complex scenarios, such as those in which background computation load is present on the resources in the Grid, and multiple WQR-FT instances are present in the same Grid. Moreover, we plan to investigate possible extensions to WQR-FT, in order

to make it able to deal with applications that require data transfers in order to carry out their computation, and with the corresponding failure models.

References

1. J.H. Abawajy. Fault-Tolerant Scheduling Policy for Grid Computing Systems. In *Proc. of 18th Int. Parallel and Distributed Processing Symposium, Workshop on*. IEEE-CS Press, April 2004.
2. F. Berman and R. Wolski et al. Adaptive Computing on the Grid Using AppLeS. *IEEE Trans. on Parallel and Distributed Systems*, 14(4), April 2004.
3. H. Casanova, F. Berman, G. Obertelli, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proc. of Supercomputing 2000*. IEEE CS Press, 2000.
4. H. Casanova, A. Legrand, and D. Zagorodnov et al. Heuristics for Scheduling Parameter Sweeping Application in Grif Environments. In *Proc. of Heterogeneous Computing Workshop*. IEEE CS Press, 2000.
5. P. Dinda D. Lu. GridG: Generating Realistic Computational Grids. *Performance Evaluation Review*, 30, 2003.
6. D.P. da Silva, W. Cirne, and F.V. Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In *Proc. of EuroPar 2003*, volume 2790 of *Lecture Notes in Computer Science*, 2003.
7. J. Brevik, D. Nurmi, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. Technical Report 37, Department of Computer Science, University of California, Santa Barbara, 2003.
8. J. Brevik, D. Nurmi, and R. Wolski. Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-peer Systems. In *Proc. of 4^{t}h Int. Workshop on Global and Peer-to-Peer Computing*, Chicago, Illinois (USA), April 19-22, 2004. IEEE Press.
9. R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauvé. Fault in Grids: Why are they so bad and What can be done about it? In *Proc. 4th Int. Workshop on Grid Computing (Grid 2003)*. IEEE-CS Press, Nov. 2003.
10. H. Schwetman. Object-oriented simulation modeling with c++/csim. In *Proc. of 1995 Winter Simulation Conference*, Dec. 1995.
11. W. Cirne, et al. Grid Computing for Bag of Tasks Applications. In *Proc. of 3rd IFIP Conf. on E-Commerce, E-Business and E-Government*, Sao Paulo, Brazil, Sept. 2003.
12. J. Weissman and D. Womack. Fault Tolerant Scheduling in Distributed Networks. Technical Report TR CS-96-10, Department of Computer Science, University of Texas, San Antonio, Sept. 1996.
13. J.W. Young. A First-order Approximation to the Optimum Checkpoint. *Communications of the ACM*, 17, 1974.
14. S. Hwang, C. Kesselman. A Flexible Framework for Fault Tolerance in the Grid. *Journal of Grid Computing*, 1(3), 2003.
15. X. Zhang, D. Zagorodnov, M. Hiltunen, K. Marzullo, and R.D. Schlichting. Fault-tolerant Grid Services Using Primary-Backup: Feasibility and Performance. In *Proc. IEEE Int. Conf. on Cluster Computing*. IEEE-CS Press, Sep. 2004.

The Design and Implementation of the KOALA Co-allocating Grid Scheduler

H.H. Mohamed and D.H.J. Epema

Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology,
P.O. Box 5031, 2600 GA Delft, The Netherlands
{H.H.Mohamed, D.H.J.Epema}@ewi.tudelft.nl

Abstract. In multicluster systems, and more generally, in grids, jobs may require *co-allocation*, i.e., the simultaneous allocation of resources such as processors and input files in multiple clusters. While such jobs may have reduced runtimes because they have access to more resources, waiting for processors in multiple clusters and for the input files to become available in the right locations, may introduce inefficiencies. In this paper we present the design of KOALA, a prototype for processor and data co-allocation that tries to minimize these inefficiencies through the use of its Close-to-Files placement policy and its Incremental Claiming Policy. The latter policy tries to solve the problem of a lack of support for reservation by local resource managers.

1 Introduction

Grids offer the promise of transparent access to large collections of resources for applications demanding many processors and access to huge data sets. In fact, the needs of a single application may exceed the capacity available in each of the subsystems making up a grid, and so *co-allocation*, i.e., the simultaneous access to resources of possibly multiple types in multiple locations, managed by different resource managers [1], may be required.

Even though multiclusters and grids offer very large amounts of resources, to date most applications submitted to such systems run in single subsystems managed by a single scheduler. With this approach, grids are in fact used as big load balancing devices, and the function of a grid scheduler amounts to choosing a suitable subsystem for every application. The real challenge in resource management in grids lies in co-allocation. Indeed, the feasibility of running parallel applications in multicluster systems by employing processor co-allocation has been demonstrated [2, 3].

In this paper, we present the design and implementation of a grid scheduler named KOALA on our wide-area Distributed ASCI Supercomputer (DAS, see Section 2.1). KOALA includes mechanisms and policies for both processor and data co-allocation in multicluster systems, and more generally, in grids. KOALA uses the Close-to-Files (CF) and the Worst Fit (WF) policy for placing job com-

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 640–650, 2005.

ponents on clusters with enough idle processors. Our biggest problem in processor co-allocation is the lack of a reservation mechanism in the local resource managers. In order to solve this problem, we propose the Incremental Claiming Policy (ICP), which optimistically postpones the claiming of the processors for the job to a time close to the estimated job start time.

In this paper we present complete design of KOALA which has been implemented and tested extensively in the DAS testbed. A more extensive description and evaluation of its placement policies can be found in [4]. The main contributions of this paper are a reliably working prototype for co-allocation and the ICP policy, which is a workaround method for processor reservation. The evaluation of the performance of ICP will be the subject of a future paper.

2 A Model for Co-allocation

In this section, we present our model of co-allocation in multiclusters and in grids.

2.1 System Model

Our system model is inspired by DAS [5] which is a wide-area computer system consisting of five clusters (one at each of five universities in the Netherlands, amongst which Delft) of dual-processor Pentium-based nodes, one with 72, the other four with 32 nodes each. The clusters are interconnected by the Dutch university backbone (100 Mbit/s), while for local communications inside the clusters Myrinet LANs are used (1200 Mbit/s). The system was designed for research on parallel and distributed computing. On single DAS clusters, PBS [6] is used as a local resource manager. Each DAS cluster has its own separate file system, and therefore, in principle, files have to be moved explicitly between users' working spaces in different clusters.

We assume a multicluster environment with sites that each contain computational resources (processors), a file server, and a local resource manager. The sites may combine their resources to be managed by a *grid scheduler* when executing jobs in a grid. The sites where the components of a job run are called its *execution sites*, and the site(s) where its input file(s) reside are its *file sites*. In this paper, we assume a single central grid scheduler, and the site where it runs is called the *submission site*. Of course, we are aware of the drawbacks of a single central submission site and currently we are working on extending our model to multiple submission sites.

2.2 Job Model

By a job we mean a *parallel application* requiring files and processors that can be split up into several *job components* which can be scheduled to execute on multiple execution sites simultaneously (*co-allocation*) [7, 8, 1, 4]. This allows the execution of large parallel applications requiring more processors than available on a single site [4]. Job requests are supposed to be *unordered*, meaning that

a job only specifies the numbers of processors needed by its components, but not the sites where these components should run. It is the task of the grid scheduler to determine in which cluster each job component should run, to move the executables as well as the input files to those clusters before the job starts, and to start the job components simultaneously.

We consider two priority levels, high and low, for jobs. The priority levels play part only when a job of a high priority is about to start executing. At this time and at the execution sites of the job, it is possible for processors requested by the job to be occupied by other jobs. Then, if not enough processors are available, a job of high priority may preempt low-priority jobs until enough idle processors for it to execute are freed (see Section 4.2).

We assume that the input of a whole job is a single data file. We deal with two models of file distribution to the job components. In the first model, job components work on different chunks of the same data file, which has been partitioned as requested by the components. In the second model, the input to each of the job components is the whole data file. The input data files have unique logical names and are stored and possibly replicated at different sites. We assume that there is a replica manager that maps the logical file names specified by jobs onto their physical location(s).

2.3 Processor Reservation

In order to achieve co-allocation for a job, we need to guarantee the simultaneous availability of sufficient numbers of idle processors to be used by the job at multiple sites. The most straight-forward strategy to do so is to reserve processors at each of the selected sites. If the local schedulers do support reservations, this strategy can be implemented by having a global grid scheduler obtain a list of available time slots from each local scheduler, and reserve a common timeslot for all job components. Unfortunately, a reservation-based strategy in grids is currently limited due to the fact that only few local resource managers support reservations (for instance, PBS-pro [9] and Maui [10] do). In the absence of processor reservation, good alternatives are required in order to achieve co-allocation.

3 The Processor and Data Co-allocator

We have developed a Processor and Data Co-Allocator (KOALA) prototype of a co-allocation service in our DAS system (see Section 2.1). In this section we describe the components of the KOALA, and how they work together to achieve co-allocation.

3.1 The KOALA Components

The KOALA consists of the following four components: the *Co-allocator* (CO), the *Information Service* (IS), the *Job Dispatcher* (JD), and the *Data Mover* (DM). The components and interactions between them are illustrated in Figure 1, and are described below.

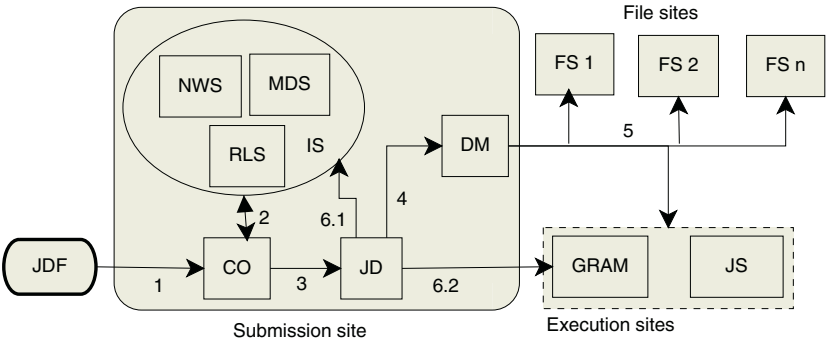


Fig. 1. The interaction between the KOALA components. The arrows correspond to the description in Section 3

The CO accepts a job r equest (arrow 1 in the figure) in the form of a job Description File (JDF). We use the Globus Resource Specification Language (RSL) [11] for JDFs, with the RSL "+"-construct to aggregate the components' requests into a single multi-request. The CO uses a placement policy (see Section 4.1) to try to place jobs, based on information obtained from the IS (arrow 2). The IS is comprised of the Globus Toolkit's Metacomputing Directory Service (MDS) [11] and Replica Location Service (RLS) [11], and *Iperf* [12], a tool to measure network bandwidth. The MDS provides the information about the numbers of processors currently used and the RLS provides the mapping information from the logical names of files to their physical locations. After a job has been successfully placed, i.e., the file sites and the execution sites of the job components have been determined, the CO forwards the job to the JD (arrow 3).

On receipt of the job, the JD instructs the DM (arrow 4) to initiate the third-party file transfers from the file sites to the execution sites of the job components (arrows 5). The DM uses Globus GridFTP [13] to move files to their destinations. The JD then determines the Job Start Time and the appropriate time that the processors required by a job can be claimed (Job Claiming Time)(Section 3.3) At this time, the JD uses a claiming policy (see Section 4.2) to determine the components that can be started based on the information from the IS (arrow 6.1). The components which can be started are sent to their Local Schedulers (LSs) of respective execution sites through the Globus Resource Allocation Manager (GRAM) [11].

Synchronization of the start of the job components is achieved through a piece of code added to the application which delays the execution of the job components until the estimated Job Start Time (see Section 4).

3.2 The Placement Queue

When a job is submitted to the system, the KOALA tries to place it according to one of its placement policies (Section 4.1). If a *placement try* fails, the KOALA

adds the job to the tail of the so-called *placement queue* , which holds all jobs that have not yet been successfully placed. The KOALA regularly scans the placement queue from head to tail to see whether any job in it can be placed. For each job in the queue we maintain its number of placement tries, and when this number exceeds a threshold, the job submission fails. This threshold can be set to ∞, i.e., no job placement fails. The time between successive scans of the placement queue is adaptive; it is computed as the product of the average number of placement tries of the jobs in the queue and a fixed interval (which is a parameter of the KOALA). The time when job placement succeeds is called its Job Placement Time (see Figure 2).

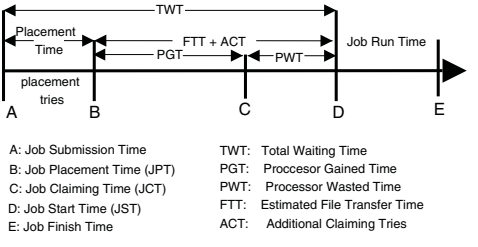


Fig. 2. The timeline of a job submission

3.3 The Claiming Queue

After the successful placement of a job, its File Transfer Time (FTT) and its Job Start Time (JST) are estimated before the job is added to the so-called *claiming queue*. This queue holds jobs which have been placed but for which the allocated processors have not yet been claimed. The job's FTT is calculated as the maximum of all of its components' estimated transfer times, and the JST is estimated as the sum of its Job Placement Time (JPT) and its FTT (see Figure 2). We then set its Job Claiming Time (JCT) (point C in Figure 2) initially to the sum of its JPT and the product of L and FTT:

$$JCT_0 = JPT + L \cdot FTT,$$

where L is job-dependent parameter, with $0 < L < 1$. In the claiming queue, jobs are arranged in increasing order of their JCT.

We try to claim (*claiming try*) at the current JCT by using our Incremental Claiming Policy (see Section 4.2). The job is removed from the claiming queue if claiming for all of its components has succeeded. Otherwise, we perform successive claiming tries. For each such try we recalculate the JCT by adding to the current JCT the product of L and the time remaining until the JST (time between points C and D in Figure 2):

$$JCT_{n+1} = JCT_n + L \cdot (JST - JCT_n).$$

If the job's JCT_{n+1} reaches its JST and for some of its components claiming has still not succeeded, the job is returned to the placement queue (see Section 3.2). Before doing so, its parameter L is decreased by a fixed amount and its components that were successfully started in previous claiming tries are aborted. The parameter L is decreased in each claiming try until its lower bound is reached so as to increase the chance of claiming success. If the number of times we have performed claiming try for the job exceeds some threshold (which can be set to ∞), the job submission fails.

We call the time between the JPT of a job and the time of successfully claiming processors for it, the Processor Gained Time (PGT) of the job. The time between the successful claiming and the actual job start time is Processor Wasted Time (PWT) (see Figure 2). During the PGT, jobs submitted through other schedulers than our grid scheduler can use the processors. The time from the submission of the job until its actual start time is called the Total Waiting Time (TWT) of the job.

4 Co-allocation Policies

In this section, we present the co-allocation policies for placing jobs and claiming processors that are used with KOALA.

4.1 The Close-to-Files Placement Policy

Placing a job in a multicluster means finding a suitable set of execution sites for all of its components and suitable file sites for the input file. (Different components may get the input file from different locations.) The most important consideration here is of course finding execution sites with enough processors. However, when there is a choice among execution sites for a job component, we choose the site such that the (estimated) delay of transferring the input file to the execution site is minimal. We call the placement policy doing just this the Close-to-Files (CF) policy. A more extensive description and performance analysis of this policy can be found in [4].

Built into the KOALA is also the Worst Fit (WF) placement policy. WF places the job components in decreasing order of their sizes on the execution sites with the largest (remaining) number of idle processors. In case the files are replicated, we select for each component the replica with the minimum estimated file transfer time to that component's execution site.

Note that both CF and WF may place multiple job components on the same cluster. We also remark that both CF and WF make perfect sense in the absence of co-allocation.

4.2 The Incremental Claiming Policy

Claiming processors for job components starts at a job's initial JCT and is repeated at subsequent claiming tries. Claiming for a component will only succeed

if there are still enough idle processors to run it. Since we want the job to start with minimal delay, the component may, in the process of the claiming policy, be re-placed using our placement policy. A re-placement can be accepted if the file from the new file site can be transferred to the new execution site before the job's JST. We can further minimize the delay of starting high priority jobs by allowing them to preempt low priority jobs at their execution sites.

We call the policy doing all of this the Incremental Claiming Policy (ICP), which operates as follows (the line numbers mentioned below refer to Algorithm 1.). For a job, ICP first determines the sets C_{prev}, C_{now}, and C_{not} of components that have been previously started, of components that can be started now based on the current number of idle processors, and of components that cannot be started based on these numbers, respectively. It further calculates F, which is the sum of the fractions of the job components that have previously been started and components that can be started in the current claiming try (line 1). We define T as the required lower bound of F; the job is returned to the claiming queue if its F is lower than T (line 2).

Algorithm 1. Pseudo-code of the Incremental Claiming Policy

Require: job J is already placed
Require: set C_{prev} of previously started components of J
Require: set C_{now} of components of J that can be started now
Require: set C_{not} of components of J that cannot be started now
1: $F \Leftarrow (|C_{prev}| + |C_{now}|)/|J|$
2: **if** $F \geq T$ **then**
3: **if** $C_{not} \neq \emptyset$ **then**
4: **for all** $k \in C_{not}$ **do**
5: $(E_k, F_k, ftt_k) \Leftarrow Place(k)$
6: **if** $ftt_k + JCT < JST$ **then**
7: $C_{now} \Leftarrow C_{not} \setminus \{k\}$
8: **else if** J priority is high **then**
9: $P_k \Leftarrow count(processors)$ * used by low-priority jobs at E_k *\
10: **if** $P_k \geq$ size of k **then**
11: **repeat**
12: Preempt low-priority jobs at E_k
13: **until** *count*(freed processors) $\geq$ size of k * at E_k *\
14: $C_{now} \Leftarrow C_{not} \setminus \{k\}$
15: start components in C_{now}

Otherwise, for each component k that cannot be started, ICP first tries to find a new pair of execution site-file site with the CF policy (line 5). On success, the new execution site E_k, file site F_k and the new estimated transfer time between them, ftt_k, are returned. If it is possible to transfer file between these sites before JST (line 6), the component k is moved from the set C_{not} to the set C_{now} (line 7).

For a job of high priority, if the file cannot be transferred before JST or the re-placement of component failed (line 8), the policy performs the follow-

ing. At the execution site E_k of component k, it checks whether the sum of its number of idle processors and the number of processors currently being used by low-priority jobs is at least equal to the number of processors the component requests (lines 9 and 10). If so, the policy preempts low-priority jobs in descending order of their JST (newest-job-first) until a sufficient number of processors have been freed (lines 11-13). The preempted jobs are then returned to the placement queue.

Finally, those components that can be claimed at this claiming try are started (line 15). For this purpose, a small piece of code has been added with the sole purpose of delaying the execution of the job barrier until the job start time. Synchronization is achieved by making each component wait on the barrier until it hears from all the other components.

When T is set to 1 the claiming process becomes atomic, i.e., claiming only succeeds if for all the job components processors can be claimed.

4.3 Experiences

We have gathered extensive experience with the KOALA while performing hundreds of experiments to asses the performance of the CF placement policy in [4]. In each of these experiments, more than 500 jobs were successfully submitted to the KOALA. These experiments proved the reliability of the KOALA. An attempt was also made to submit jobs to the GridLab [14] testbed, which is a heterogeneous grid environment. KOALA managed also to submit jobs successfully to this testbed, and currently more experiments have been planned on this testbed.

5 Related Work

In [15, 16], co-allocation (called multi-site computing there) is studied also with simulations, with as performance metric the (average weighted) response time. There, jobs only specify a total number of processors, and are split up across the clusters. The slow wide-area communication is accounted for by a factor r by which the total execution times are multiplied. Co-allocation is compared to keeping jobs local and to only sharing load among the clusters, assuming that all jobs fit in a single cluster. One of the most important findings is that for r less than or equal to 1.25, it pays to use co-allocation. In [17] an architecture for a grid superscheduler is proposed, and three job migration algorithms are simulated. However, there is no real implementation of this scheduler, and jobs are confined to run within a single subsystem of a grid, reducing the problem studied to a traditional load-balancing problem.

In [18], the Condor class-ad matchmaking mechanism for matching single jobs with single machines is extended to "gangmatching" for co-allocation. The running example in [18] is the inclusion of a software license in a match of a job and a machine, but it seems that the gangmatching mechanism might be extended to the co-allocation of processors and data.

5by binding execution and storage sites into I/O communities that reflect the physical reality.

In [19], the scheduling of sequential jobs that need a single input file is studied in grid environments with simulations of synthetic workloads. Every site has a Local Scheduler, an External Scheduler (ES) that determines where to send locally submitted jobs, and a Data Scheduler (DS) that asynchronously, i.e., independently of the jobs being scheduled, replicates the most popular files stored locally. All combinations of four ES and three DS algorithms are studied, and it turns out that sending jobs to the sites where their input files are already present, and actively replicating popular files, performs best.

In [20], the creation of abstract workflows consisting of application components, their translation into concrete workflows, and the mapping of the latter onto grid resources is considered. These operations have been implemented using the Pegasus [21] planning tool and the Chimera [22] data definition tool. The workflows are represented by DAGs, which are actually assigned to resources using the Condor DAGMan and Condor-G [23]. As DAGs are involved, no simultaneous resource possession implemented by a co-allocation mechanism is needed.

In the AppLes project [24], each grid application is scheduled according to its own performance model. The general strategy of AppLes is to take into account resource performance estimates to generate a plan for assigning file transfers to network links and tasks (sequential jobs) to hosts.

6 Conclusions

We have addressed the problem of scheduling jobs consisting of multiple components that require both processor and data co-allocation in multicluster systems and grids in general. We have developed KOALA, a prototype for processor and data co-allocation which implements our placement and claiming policies. Our initial experiences show the correct and reliable operation of the KOALA.

As future work, we are planning to remove the bottleneck of a single global scheduler, and to allow flexible jobs that only specify the total number of processors needed and allow the KOALA to fragment jobs into components (the way of dividing the input files across the job components is then not obvious). In addition, more extensive performance study of the KOALA in a heterogeneous grid environment has been planned.

References

1. Czajkowski, K., Foster, I.T., Kesselman, C.: Resource Co-Allocation in Computational Grids. In: Proc. of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8). (1999) 219–228

2. van Nieuwpoort, R., Maassen, J., Bal, H., Kielmann, T., Veldema, R.: Wide-Area Parallel Programming Using the Remote Method Invocation Method. Concurrency: Practice and Experience **12** (2000) 643–666
3. Banen, S., Bucur, A., Epema, D.: A Measurement-Based Simulation Study of Processor Co-Allocation in Multicluster Systems. In Feitelson, D., Rudolph, L., Schwiegelshohn, U., eds.: 9th Workshop on Job Scheduling Strategies for Parallel Processing. Volume 2862 of LNCS. Springer-Verlag (2003) 105–128
4. Mohamed, H., Epema, D.: An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters. In: Proc. of CLUSTER 2004, IEEE Int'l Conference Cluster Computing 2004. (2004)
5. Web-site: (The Distributed ASCI Supercomputer (DAS)) `http://www.cs.vu.nl/das2`.
6. Web-site: (The Portable Batch System) `www.openpbs.org`.
7. Bucur, A., Epema, D.: Local versus Global Queues with Processor Co-Allocation in Multicluster Systems. In Feitelson, D., Rudolph, L., Schwiegelshohn, U., eds.: 8th Workshop on Job Scheduling Strategies for Parallel Processing. Volume 2537 of LNCS. Springer-Verlag (2002) 184–204
8. Ananad, S., Yoginath, S., von Laszewski, G., Alunkal, B.: Flow-based Multistage Co-allocation Service. In d'Auriol, B.J., ed.: Proc. of the International Conference on Communications in Computing, Las Vegas, CSREA Press (2003) 24–30
9. Web-site: (The Portable Batch System) `http://www.pbspro.com/`.
10. Web-site: (Maui Scheduler) `http://supercluster.org/maui/`.
11. Web-site: (The Globus Toolkit) `http://www.globus.org/`.
12. Web-site: (Iperf Version 1.7.0) `http://dast.nlanr.net/Projects/Iperf/`.
13. Allcock, W., Bresnahan, J., Foster, I., Liming, L., Link, J., Plaszczac, P.: GridFTP Update. Technical report (2002)
14. Web-site: (A Grid Application Toolkit and Testbed) `http://www.gridlab.org/`.
15. Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., Streit, A.: On Advantages of Grid Computing for Parallel Job Scheduling. In: 2nd IEEE/ACM Int'l Symposium on Cluster Computing and the GRID (CCGrid2002). (2002) 39–46
16. Ernemann, C., Hamscher, V., Streit, A., Yahyapour, R.: Enhanced Algorithms for Multi-Site Scheduling. In: 3rd Int'l Workshop on Grid Computing. (2002) 219–231
17. Shan, H., Oliker, L., Biswas, R.: Job superscheduler architecture and performance in computational grid environments. In: Supercomputing '03. (2003)
18. Raman, R., Livny, M., Solomon, M.: Policy driven heterogeneous resource co-allocation with gangmatching. In: 12th IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-12). IEEE Computer Society Press (2003) 80–89
19. Ranganathan, K., Foster, I.: Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In: 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 Edinburgh, Scotland. (2002)
20. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K.: Mapping Abstract Complex Workflows onto Grid Environments. J. of Grid Computing **1** (2003) 25–39
21. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Vahi, G.M.K., Koranda, S., Lazzarini, A., Papa, M.A.: From Metadata to Execution on the Grid Pegasus and the Pulsar Search. Technical report (2003)
22. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In: 14th Int'l Conf. on Scientific and Statistical Database Management (SSDBM 2002). (2002)

23. Frey, J., Tannenbaum, T., Foster, I., Livny, M., Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. In: Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC), San Francisco, California (2001) 7–9
24. Casanova, H., Obertelli, G., Berman, F., Wolski, R.: The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. (2000) 75–76

A Multi-agent Infrastructure and a Service Level Agreement Negotiation Protocol for Robust Scheduling in Grid Computing[1]

D. Ouelhadj[1], J. Garibaldi[2], J. MacLaren[3], R. Sakellariou[4], and K. Krishnakumar[5]

[1, 2] School of Computer Science and IT, University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK
{dxs, jmg}@cs.nott.ac.uk
[3, 4, 5] University of Manchester, Oxford Road, Manchester, M13 9PL, UK
{jon.maclaren, rizos, krish}@man.ac.uk

Abstract. In this paper we propose a new infrastructure for efficient job scheduling on the Grid using multi-agent systems and a Service Level Agreement (SLA) negotiation protocol based on the Contract Net Protocol. The agent-based Grid scheduling system involves user agents, local scheduler agents, and super scheduler agents. User agents submit jobs to Grid compute resources. Local scheduler agents schedule jobs on compute resources. Super scheduler agents act as mediators between the local scheduler and the user agents to schedule the jobs at the global level of the Grid. The SLA negotiation protocol is a hierarchical bidding mechanism involving meta-SLA negotiation between the user agents and the super scheduler agents; and sub-SLA negotiation between the super scheduler agents and the local scheduler agents. In this protocol the agents exchange SLA-announcements, SLA-bids, and SLA-awards to negotiate the schedule of jobs on Grid compute resources. In the presence of uncertainties a re-negotiation mechanism is proposed to re-negotiate the SLAs in failure.

1 Introduction

Grid computing has emerged as a new paradigm for next generation distributed computing. A Computational Grid is an infrastructure which enables the interconnection of substantial distributed compute resources in order to solve large scale problems in science, engineering and commerce that cannot be solved on a single system [6]. One of the most challenging issues from a Grid infrastructure perspective is the efficient schedule of jobs on distributed resources. Scheduling jobs in a Grid computing environment is a complex problem as resources are geographically distributed having different usage policies and may exhibit highly non-uniform performance characteristics, heterogeneous in nature, and have varying loads and availability.

[1] This work is funded by the EPSRC Fundamental Computer Science for e-Science initiative (Grants GR/S67654/01 and GR/S67661/01), whose support we are pleased to acknowledge.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 651 – 660, 2005.

The Grid scheduling problem is defined as: “given appropriate input, the high performance scheduler determines an application schedule, which is an assignment of tasks, data, and communication to resources, ordered in time, based on the rules of the scheduling policy, and evaluated as performance efficient under the criteria established by the objective function. The goal of the high-performance scheduler is to optimise the performance [1].

Within the Grid community at present, there is a keen focus on the management and scheduling of workflows, i.e. complex jobs, consisting multiple computational tasks, connected either in a Directed Acyclic Graph (DAG), or in a more general graph, incorporating conditionals and branches [15]. In order for a workflow enactment engine to successfully orchestrate these workflows, it must be possible to schedule multiple computational tasks onto distributed resources, while still respecting any dependence in the workflow. Current methods employed to schedule work on compute resources within the Grid are unsuitable for this purpose. Traditionally, these scheduling systems are queue based and use simple heuristics such as First-Come-First-Served (FCFS), or more complicated and efficient methods such as Backfilling, Gang Scheduling, Time Slicing, etc [9]. Such batch scheduling systems provide only one level of service, namely ‘run this when it gets to the head of the queue’, which approximates to ‘whenever’. New patterns of usage arising from Grid computing have resulted in the introduction of advance reservation to these schedulers, where jobs can be made to run at a precise time. However, this is also an extreme level of service, and is excessive for many workflows, where often it would be sufficient to know the latest finish time, or perhaps the soonest start time and latest end time. Advance reservation (in its current form) is also unsuitable for any scenario involving the simultaneous scheduling of multiple computation tasks, either as a sequence, or tasks that must be co-scheduled. Also, advance reservation has several disadvantages for the resource owner. When an advance reservation is made, the scheduler must place jobs around this fixed job. Typically, this is done using backfilling, which increases utilisation by searching the work queues for small jobs, which can plug the gaps. In practice, this rarely works perfectly, and so the scheduler must either leave the reserved processing elements empty for a time, or suspend or checkpoint active jobs near to the time of the reservation. Either way, there are gaps in the schedule, i.e., CPU time which is not processing users’ work. As utilisation often represents income for the service owner, there is a tendency to offset the cost of the unused time by charging for advance reservation jobs at a considerably higher tariff. As these new patterns of usage increase, utilisation will fall further. While it is possible to set tariffs high enough to compensate, this brute-force solution is inefficient in terms of resources, and undesirable for both users, who pay higher prices, and for resource owners, who must charge uncompetitive prices.

Recently, Multi-Agent Systems [5, 14] and the Contract Net Protocol [18] have proven a major success in solving dynamic scheduling problems and have given answers to the problem of how to efficiently integrate and cooperate communities of distributed and interactive systems in a wide range of applications [16]. A multi-agent system is a network of agents that work together to solve problems that are beyond their individual capabilities [14]. Multi-agent systems have been known to provide capabilities of autonomy, cooperation, heterogeneity, robustness and reactivity, scalability, and flexibility [5, 16]. A number of initiatives to apply agents in

computational Grids have appeared [2, 7, 11]. Most these agent-based Grid scheduling systems are centralised and static as scheduling is performed by a Grid high-performance scheduler (broker), and resource agents do not use any flexible negotiation to schedule the jobs. In a Grid environment resources are geographically distributed and owned by different individuals. It is not practical that a single point in the virtual system retains entire Grid's information that can be used for job scheduling and cope with the dynamic nature of the Grid environment. Therefore, the use distributed scheduling and negotiation models would be more efficient, flexible, and robust.

In this paper we propose a fundamental new infrastructure for efficient job scheduling on the Grid based on multi-agent systems, and a Service Level Agreement (SLA) negotiation protocol based on the Contract Net Protocol [13]. Section 2 describes SLAs. Section 3 presents the proposed SLA based Grid scheduling multi-agent system infrastructure. Section 4 elaborates the SLA negotiation protocol. Section 5 describes the SLA re-negotiation process in the presence of failures. Conclusions are presented in Section 6.

2 Service Level Agreement

Service Level Agreements (SLAs) [3, 10] are emerging as the standard concept by which work on the Grid can be arranged and co-ordinated. An SLA is a bilateral agreement, typically between a service provider and a service consumer. These form a natural choice for representing the agreed constraints for individual jobs. While there are technologies for composing SLAs in XML-based representations, e.g. WSLA [12], these embed domain-specific terms; no terms for resource reservation have yet been proposed within the Grid community. In any case, it is certain that SLAs can be designed to include acceptable start and end time bounds and a simple description of resource requirements. SLAs expressing conventional requirements of “at time HH:MM” or “whenever” could still be used where necessary, although these—especially the latter—may not be accepted by the scheduler. The GGF GRAAP Working Group [8] is interested in SLA terms for resource reservation, but has not yet put forward any designs for what these SLAs should look like. It is intended that the project will feed back a design of these SLA terms to the community, contributing to the standardisation process, and influencing the development of emerging technologies, such as the negotiation protocol WS-Agreement, which is being defined by the GGF GRAAP Working Group.

3 The Grid Scheduling Multi-agent System Infrastructure

The multi-agent infrastructure proposed is organised as a population of cognitive, autonomous, and heterogeneous agents, for integrating a range of scheduling objectives related to Grid compute resources. Figure 1 shows the multi-agent infrastructure for SLA based Grid scheduling. The infrastructure involves three types of agents: User agents, Local Scheduler (LS) Agents, and Super Scheduler (SS) Agents. Three databases are also used in this architecture to store information on SLAs, LS agents, and resources.

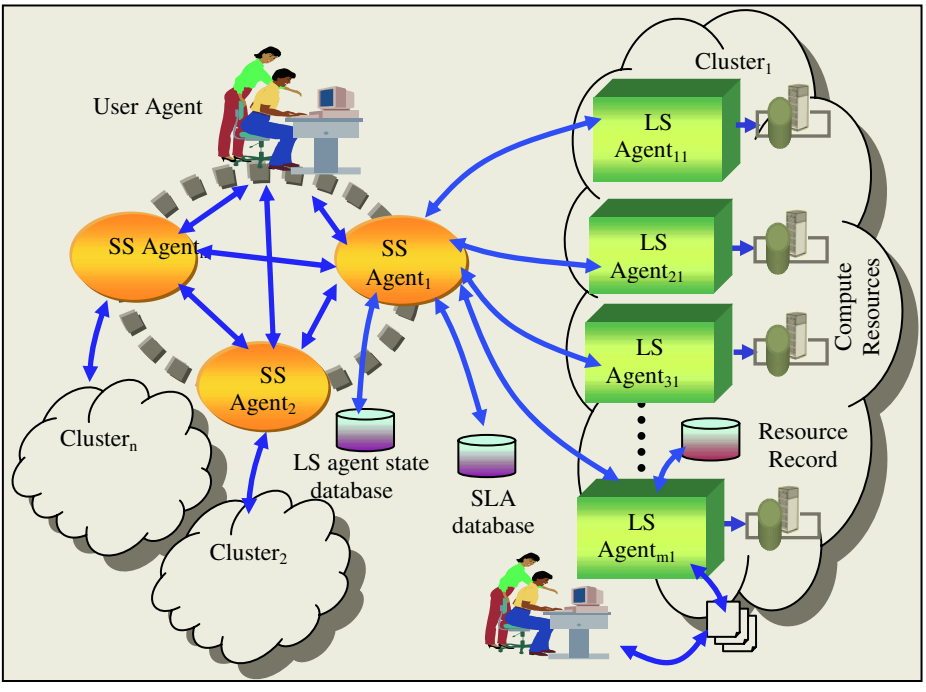


Fig. 1. SLA based grid scheduling system infrastructure

User Agent: The user agent requests the execution of jobs on the Grid and negotiates SLAs with the SS agents. The user agent can also submit jobs locally to SL agents.

Local Scheduler Agent: Each computer resource within each institution is assigned to an LS agent. The set of compute resources and their corresponding LS agents constitute a cluster. The LS agents are responsible for scheduling jobs, usually assigned by the SS agents, on compute resources within the cluster. However, jobs cannot only be arriving from the SS agents, but they can also be injected from other means such as PC which is locally connected to the machine that hosts the LS agent.

Super Scheduler Agent: SS agents are distributed in the Grid having each at every institution, and act as mediators between the user agent and the LS agents. Each cluster of LS agents of the same institution is coordinated by an SS agent. The user agent usually submits jobs to the SS agents. SS agents negotiate SLAs with the user agent and the LS agents to schedule the jobs globally on compute resources.

Databases: The databases used in the architecture are the following:

- *Resource record*: holds information about the compute resources on which the jobs are executed by an LS agent.
- *SLA database*: stores the description of the SLAs.
- *LS agent state database*: holds the status of LS agents and the loading/usage information about their local resources which is used by the SS agents.

4 SLA Negotiation Protocol

We propose an SLA negotiation protocol based on the Contract Net Protocol to allow the SS agents to negotiate with the user agent and the LS agents the schedule of jobs on the distributed computer resources. The Contract Net Protocol is a high level protocol for achieving efficient cooperation introduced by Smith [18] based on a market-like protocol. The basic metaphor used in the Contract Net Protocol is, as the name of the protocol suggests, contracting. It is the most common and best-studied mechanism for distributed task allocation in agent-based systems [5, 16]. In this protocol, a decentralised market structure is assumed and agents can take on two roles: a manager and a contractor. The manager agent advertises the task by a *task announcement* to other agents in the net. In response, contracting agents evaluate the task with respect to their abilities and engagements and submit *bids*. A bid indicates the capabilities of the bidder that are relevant to the execution of the announced task. The manager agent evaluates the submitted bids, and selects the most appropriate bidder to execute the task, which leads to *awarding* a contract to the contractor with the most appropriate bid. The advantages of the contract net protocol include the following: dynamic task allocation via self-bidding which leads to better agreements; it provides natural load balancing (as busy agents need not bid); agents can be introduced and removed dynamically; and it is a reliable mechanism for distributed control and failure recovery.

The proposed SLA negotiation protocol is a hierarchical bidding mechanism involving two negotiation levels:

1. Meta-SLA negotiation level: In this level the SS agents and the user agents negotiate meta-SLAs. A meta-SLA contains high-level description of jobs supplied by the user and may be refined or additional information may be added in the final agreed SLA at the end of negotiation. A meta-SLA may have the following baseline information:

- Resource information: the required capability of the resource for the application. It is usually very high level description of a resource metric containing resource details such as machine information, the range of processors, etc.
- Estimated due date: the time by which the job should be finished.
- Estimated cost: it is the maximum cost limit pre-set by the user for any given job execution request.
- Required execution host information (optional).

Uncertainty in user requirements such as time and cost constraint is represented by fuzzy numbers [17, 19].

2. Sub-SLA negotiation level: In this level the SS agents negotiate sub-SLAs with the LS agents. The SS agents decompose the meta-SLA into its low level resource attributes, sub-SLAs which contain low level raw resource description such as processes, memory processors, etc. A sub-SLA may have the following baseline information:

- Low level raw resource descriptions such as number of nodes, architecture, memory, disk size, CPU, network, OS, application (if any needed).
- Time constraint.
- Cost constraint.
- Storage Requirement.

4.1 Steps of the Hierarchical SLA Negotiation Protocol

The hierarchical SLA negotiation protocol involves several negotiation steps at each level of the protocol. The steps are described below and illustrated in figure 2.

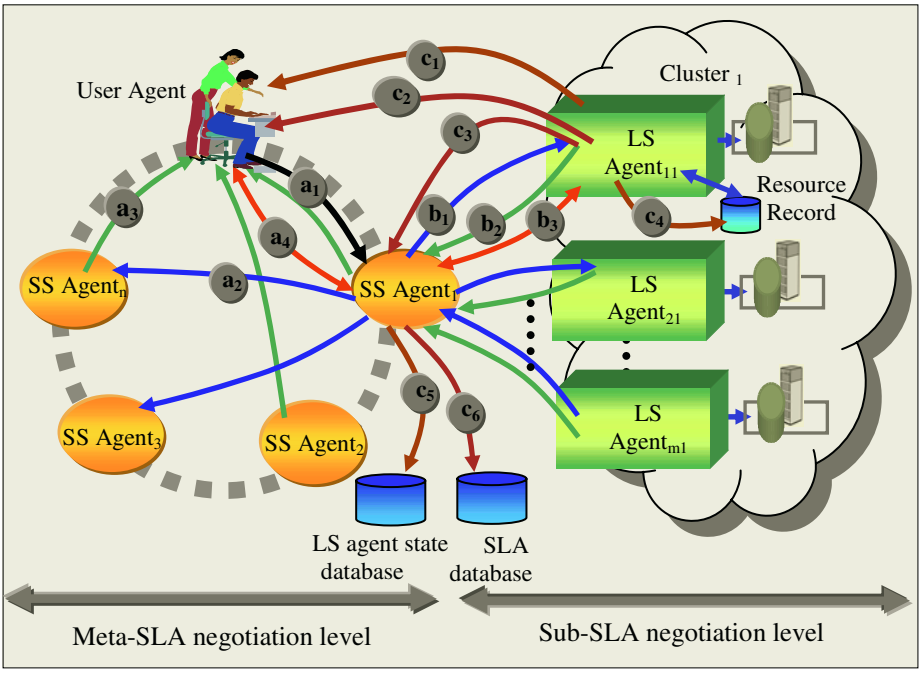


Fig. 2. Steps of the SLA negotiation protocol

a. Meta-SLA negotiation steps: Meta-SLA negotiation steps are the following:

a_1. Meta-SLA-request: The user agent submits a meta-SLA to the nearest SS agent to request the execution of a job. The meta-SLA at this stage describes baseline user requirements such as required resources over a time period, job start time and end time, the cost the user is prepared to pay, required execution host information, etc.

a_2. Meta-SLA-announcement: Upon receiving the meta-SLA request from the user agent, the SS agent advertises the meta-SLA to the neighbouring SS agents in the net by sending a meta-SLA-announcement.

a_3. Meta-SLA-bidding: In response to the meta-SLA-announcement, each SS agent queries the LS agent state database to check the availability of required resources for the job response time to identify suitable set of resources. Cost is then evaluated for each potential resource. The SS agents select the best resources which satisfy the availability and cost constraints of the job. Each SS agent submits a meta-SLA-bid to the user agent. The meta-SLA-bid describes the new estimated response time and cost, and other additional information such as proposed compensation package.

a_4. Meta-SLA-award: The user agent, upon receiving the meta-SLA-bids, selects the most appropriate SS agent with the best meta-SLA-bid and sends a notification agreement to its. The selected SS agent stores the agreed meta-SLA in the SLAs database. The user has now an agreement with the Grid provider to use its resources.

b. Sub-SLA negotiation steps: Once the SS agent has been awarded the agreement, it decomposes the meta-SLA by generating one or more sub-SLAs and negotiates the sub-SLAs with the local LS agents. The sub-SLA negotiation steps are the following:

b_1. Sub-SLA-announcement: The SS agent sends the sub-SLA-announcements to the identified suitable LS agents within the cluster under its control.

b_2. Sub-SLA-bidding: The LS agents, upon receiving the sub-SLA-announce ments, query the resource record for information on CPU speed, memory size, etc. to identify suitable resources and submit bids to the SS agent.

b_3. Sub-SLA-award: The SS agent evaluates the best sub-SLA in accordance to time and cost criteria and sends a notification agreement to the most appropriate LS agent with the best resources. The SS agent stores the sub-SLA in the SLAs database and updates the LS agent state information. The selected LS agents update the resource record and reserve the resources to execute the job.

c. Task execution steps: The basic stages for task execution are the following:

c_1. The LS agent sends a notification of initiation of job execution to the user agent.
c_2. At the end of job execution, a final report including the output file details is sent to the user agent.
c_3. SL agent sends a notification end job execution to the SS agent.
c_4. SL agent updates information held in the resource record database.
c_5. SS agent updates the LS agent state database.
c_6. SS agent updates the status of the SLAs in the SLAs database.

5 SLA Re-negotiation in the Presence of Uncertainties

Re-negotiation has been identified as a long-term goal of the RealityGrid UK e-Science Pilot Project [4]; here, simulations may be collaboratively visualized and steered as the simulation runs. Even once a SLA has been agreed, re-negotiation is required for multiple reasons: to extend the running time of an increasingly interesting simulation; to increase the computational resources dedicated to either the simulation, thereby accelerating the experiment, or to the visualization, in order to improve the resolution of the rendering; resources might also be given back when the improved speed or picture quality was no longer required. Also, more generally, resources may

fail unpredictably, high-priority jobs may be submitted, etc. In a busy Grid environment, SLAs would be constantly being added, altered or withdrawn, and hence scheduling would need to be a continual, dynamic and uncertain process. The introduction of re-negotiation, permitted during job execution (as well as before it commences) makes the schedule more dynamic, requiring more frequent rebuilding of the schedule.

In the presence of failures, the SS agents re-negotiate the SLAs in failure at the local and global levels in order to find alternative LS agents to execute the jobs in failure. The basic steps of failure recovery are the following, as shown in figure 3:

f_1. LS $agent_{11}$ notifies SS $agent_1$ of the failure.
f_2. SS $agent_1$ re-negotiates the sub-SLAs in failure to find alternative LS agents locally within the same cluster by initiating a sub-SLA negotiation session with the suitable LS agents.
f_3. If it cannot manage to do so, the SS $agent_1$ re-negotiates the meta-SLAs with the neighbouring SS agents by initiating a meta-SLA negotiation session.
f_4. SS $agent_{2, \ldots,}$ SS $agent_n$ re-negotiate the sub-SLAs in failure to find alternative LS agents.
f_5. SS $agent_2$ located LS $agent_{22}$ to execute the job in failure.
f_6. At the end of task execution, LS $agent_{22}$ sends a final report including the output file details to the user agent.
f_7. In case the SS $agent_1$ could not find alternative LS agents at the local and global levels, the SS $agent_1$ sends an alert message to the user agent to inform him that the meta-SLA cannot be fulfilled and the penalty fee needs to be paid.

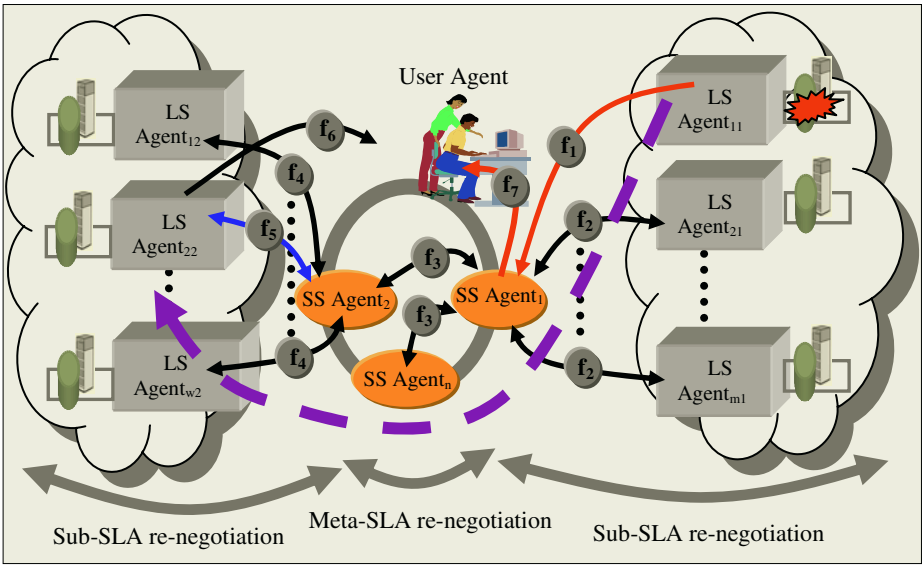


Fig. 3. Re-negotiation in the presence of resource failure

6 Conclusion

In this paper we have proposed a new infrastructure for efficient job scheduling on the Grid using multi-agent systems and a SLA negotiation protocol based on the Contract Net Protocol. Local autonomy and cooperation capabilities of the multi-agent architecture offer prospects for the most challenging issues in Grid scheduling which are: integration, heterogeneity, high flexibility, high robustness, reliability, and scalability. The SLA negotiation protocol proposed based on the Contract Net Protocol allows the user agent a flexible and robust negotiation of the execution of jobs on the Grid. The SLA negotiation protocol is a hierarchical bidding mechanism which yields many advantages relevant to Grid scheduling, such as dynamic task allocation, natural load-balancing as busy agents do not need to bid, dynamic introduction and removal of resources, and robustness against failures. To deal with the presence of uncertainties, re-negotiation is proposed to allow the agents to re-negotiate the SLAs in failure.

References

1. Berman, F.: High performance schedulers. In: Foster, I., Kesselman, C. (Eds.): The Grid: Blueprint for a new computing infrastructure. Morgan Kaufman Publishers (1998) 279-307.
2. Cao, J., Jarvis, S.: ARMS: An agent-based resource management system for Grid computing. Scientific Programming, 10 (2002) 135-148.
3. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A Protocol for negotiating service level agreements and coordinating resource management in distributed systems. Lecture Notes in Computer Science, Vol. 2537 (2002) 153-183.
4. Czajkowski, K., Pickles, S. Pruyne, J., Sander, V.: Usage scenarios for a Grid resource allocation agreement protocol. Draft Global Grid Forum Informational Document (2003).
5. Ferber, J. (ed.): Multi-agent systems: An introduction to Distributed Artificial Intelligence. Addison-Wesley, London (1999).
6. Foster, I. and Kesselman, C. (eds.): The Grid: Blueprint for a new computing infrastructure. Morgan Kaufman Publishers (1998).
7. Frey, J., Tannenbaum, T., Livny, M.: Condor-G: a computational management agent for multi-institutional Grid. Cluster Computing, 5 (2002) 237-246.
8. GRAAP: GRAAP-WG, Grid resource allocation agreement protocol working group in the Global Grid Forum. Website: https://forge.gridforum.org/projects/graap-wg/ (2004).
9. Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R.: Evaluation of job scheduling strategies for Grid computing. Lecture Notes in Computer Science (2000) 191-202.
10. Keller, A., Kar, G., Ludwig, H., Dan, A., Hellerstein, J.L.: Managing dynamic services: A contract based approach to a conceptual architecture. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (2002) 513–528.
11. Krauter, K., Buyya, R., Maheswaran, M. A taxonomy and survey of Grid resource management systems. Software Practices Experience, 32 (20020 135-164.
12. Ludwig, H., Keller, A., Dan, A., King, R.: A service level agreement language for dynamic electronic services. Proceedings of the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (2002) 25–32.
13. MacLaren, J., Sakellariou, R., Garibaldi, J., Ouelhadj, D.: Towards service level agreement based scheduling on the Grid. Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services, in the 14th International Conference on Automated Planning & Scheduling, Whistler, Canada (2004) 100-102.

14. O'Hare, G., Jennings, N. (Eds.): Foundations of Distributed Artificial Intelligence, Wiley, New York (1996).
15. Sakellariou, R., and Zhao, H.: A hybrid heuristic for DAG scheduling on heterogeneous systems. Proceedings of the 13th International Heterogeneous Computing Workshop (2004).
16. Shen, W., Norrie, D., Barthes, J. (eds.): Multi-agent systems for concurrent intelligent design and manufacturing, Taylor & Francis, London (2001).
17. Slowinski, R., and Hapke, M. (eds.): Scheduling under fuzziness. Physica Verlag (2000).
18. Smith, R.: The contract net protocol: high level communication and control in distributed problem solver. IEEE Transactions on Computers, 29 (1980) 1104-1113.
19. Zadeh, L.A.: Fuzzy Sets. Information and Control, 8 (1965) 338-353.

Towards Quality of Service Support for Grid Workflows

Ivona Brandic, Siegfried Benkner, Gerhard Engelbrecht, and Rainer Schmidt

Institute for Software Science, University of Vienna,
Nordbergstrasse 15, A-1090 Vienna, Austria
brandic@par.univie.ac.at

Abstract. Service-oriented workflow languages are being considered as a key programming paradigm for composing Grid applications from basic services. In this paper we present QoS support for grid workflows addressing the special requirements of time-critical Grid applications, as for example medical simulation services. QoS support for grid workflow is currently being developed in the context of the Vienna Grid Environment, a service-oriented Grid infrastructure for the provision of parallel simulation codes as QoS-aware Grid services, which are capable of dynamically negotiating with clients various QoS guarantees, e.g. with respect to execution time and price. We use a real world Grid service for medical image reconstruction to describe the main features of our approach and present the first prototype of a QoS-aware workflow engine which supports dynamic QoS negotiation and QoS-aware workflow execution.

1 Introduction

Service-oriented workflow languages are considered as an important tool for composing advanced Grid applications from basic Grid services and for constructing higher-level composite Grid services. Basic parts of a Grid workflow are activities which usually represent invocations of Grid service operations. Dependencies between activities are expressed in terms of control and data flow. Workflows can be composed using an algebraic approach [19] or a more sophisticated graph-based approach [13]. Graph-based workflows allow the synchronization between concurrently executed activities whereas algebraic workflows permit only a structured way of programming.

The recent shift of Grid technologies towards the adoption of standard Web Services has motivated several projects which are developing Web Service-based workflow languages and engines including BPEL [1], Triana [14], Freefluo [16] and several others.

The acceptance of Grid technologies by users depends on the ability of Grid applications to satisfy users' needs. For example, in application areas like medical simulation for surgical procedures a major requirement are execution time guarantees for time critical applications. In the context of the EU project GEMSS [6], which develops a test bed for GRID-enabled medical application services, we

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 661–670, 2005.

have designed and implemented a QoS-enabled grid service provision framework [5]. Using such a framework native HPC applications can be exposed as QoS-aware Grid services which enable clients to negotiate dynamically QoS guarantees with respect to execution time, price and other QoS properties. Dynamic composition of QoS-aware services via a workflow structure allows the customization of services and the automation of medical processes according to the concrete medical problem. Although many groups are currently extending workflow systems for Grid computing, our work on QoS-aware workflow is the first attempt to provide application-level QoS support for Grid workflows.

The major contributions of this paper are QoS extensions of the Business Process Execution Language (BPEL) and the development of a corresponding QoS-aware workflow execution engine which supports dynamic service selection, and QoS negotiation in order to address the requirements of time critical applications. The language extensions for QoS-aware workflows allow the user to associate each activity of a workflow with certain QoS constraints. Based on these constraints, our QoS-aware workflow execution engine negotiates with multiple potential service providers and dynamically selects services which can fulfill the specified QoS constraints. Furthermore our execution engine is capable of determining QoS constraints for composite services by accumulating the constraints of basic services. In order to accomplish these tasks we exploit the QoS-awareness of Grid services as provided by the Vienna Grid Environment (VGE) [5].

The remainder of this paper is structured as follows: Section 2 gives an overview about the Vienna Grid Environment, a QoS-aware grid service provision framework. Section 3 describes QoS extensions of BPEL using a medical image reconstruction workflow as example. Section 4 presents our first prototype of a QoS-aware workflow execution engine and discusses the lifecycle of a typical QoS-aware workflow. Section 5 gives an overview about related work. Conclusions and an outlook to future work are presented in Section 6.

2 QoS-Aware Grid Services

A major prerequisite for the development of QoS-aware workflows are services, which can provide QoS guarantees, for example with respect to execution time. The Vienna Grid Environment (VGE) [5], a service-oriented infrastructure for the provision of HPC applications as Grid Services, supports a flexible QoS negotiation model where clients may negotiate dynamically and on a case-by-case basis QoS guarantees on execution time, price, and others with potential service providers. VGE services encapsulate native HPC applications and offer a set of common operations for job execution, job monitoring, data staging, error recovery, and application-level quality of service support. VGE services are exposed using WSDL [9] and securely accessed via SOAP/WS-Security [18, 21].

In order to provide support for dynamic QoS negotiation, VGE services rely on a generic QoS module which comprises an *application-specific performance model*, a *pricing model*, a *compute resource manager* and several XML descriptors (see Figure 1 (a)).

Prior to QoS negotiation the desired QoS constraints have to be specified by the client in a *QoS descriptor*. Furthermore, the client has to provide a set of request parameters with meta-data about a specific service request in a *request descriptor*. For example, in the case of an image reconstruction service, request parameters typically include image size and required accuracy (i.e. number of iterations). A *machine descriptor*, supplied by the service provider, specifies the resources (number of CPUs, main memory, disk space, etc.) that can be offered for an application service.

At the center of the QoS module is an application-specific performance model which takes as input a request descriptor and a machine descriptor and returns a *performance descriptor* which includes the estimated execution time for a specific service request. The QoS management module then relies on heuristics that consider the outcome of the performance model, the availability of resources via the compute resource manager, and the service provider's pricing model to decide whether the client's QoS constraints can be fulfilled.

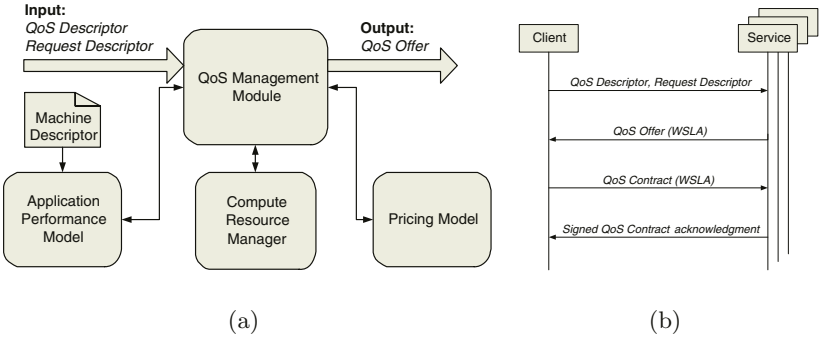


Fig. 1. QoS negotiation module (a) QoS basic scenario (b)

The basic QoS negotiation scenario as shown in Figure 1 (b) is based on a request/offer model where clients request offers from services. A client initiates the negotiation by supplying a QoS descriptor and a request descriptor. If a service decides to make an appropriate QoS offer, a temporary resource reservation with a short expiration time is made by the compute resource manager. Only if a QoS offer is confirmed by the client, a QoS contract in terms of a Web Service Level Agreement (WSLA) [20] is established and signed by both parties.

The VGE service provision framework is currently utilized in the context of the EU Project GEMSS [12] which focuses on the provision of advanced medical simulation services via a Grid infrastructure [6].

3 Workflow Language Extensions for QoS Support

The first objective of QoS-aware workflow support is to enable the specification of QoS constraints for services. The second objective is to compute QoS constraints

for the whole workflow based on the constraints of basic activities. In order to achieve these goals we extend BPEL in order to allow the specification of QoS constraints as described next.

3.1 BPEL Subset

In this section we describe the BPEL subset used to define QoS-aware workflows. BPEL is an XML-based language that uses WSDL [9] documents in order to invoke Web Services. The basic construct of the language is an `activity`. BPEL distinguishes between basic and complex activities. Basic activities perform primitive actions such as invoking other services with the `<invoke>` activity, receiving a request from a client with the `<receive>` activity or sending a reply to a client with the `<reply>` activity. Complex activities group basic or other complex activities defining a special behavior of that group of services. The `<sequence>` activity executes activities in the given order, whereas the `<flow>` activity concurrently executes underlying activities. The `<variable>` activity may be used to define input and output variables, which can be message parts defined in a WSDL document or simple data types. Variables can be copied using a `<copy>` activity and `<from>` and `<to>` activities.

3.2 Medical Image Workflow Example

One of the applications utilized within the GEMSS project is a Medical Image Reconstruction Service, applying a compute intensive fully 3D ML-EM reconstruction of 2D projection data recorded by a Single Photon Emission Computed Tomography (SPECT) [4]. A client GUI utilized for the SPECT service implements the basic invocation scenario of the SPECT service which consists of the upload of 2D projection data, the start of the reconstruction, and finally the download of the reconstructed 3D images. These basic steps are currently controlled by the user via the GUI. The execution of the SPECT service methods can be automated using a workflow language and a workflow execution engine. A corresponding BPEL workflow [1] is presented in Figure 2.

First, the input data is uploaded to the service using the `invoke` activity named `upload`. Second, the `start` activity initiates the start of the reconstruction. Finally, the reconstructed 3D image is downloaded. The workflow shown in Figure 2 specifies only the order of execution of activities.

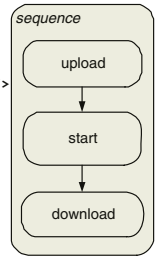
3.3 BPEL Extensions for the Specification of QoS Constraints

In order to specify QoS constraints in BPEL, we introduce the `<qos-constraints>` element, which can be used within the `<sequence>`, `<flow>` and `<invoke>` activities. Figure 3 shows an image reconstruction workflow with a corresponding `<qos-constraints>` element.

[1] Instead of `PartnerLinks` we use the `wsdl` attribute for locating external services.

```
...
<sequence name="SPECTSequence">
  <invoke name="upload"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="upload" inputVariable="uploadData"/>
  <invoke name="start"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="start" inputVariable="Cid"/>
  <invoke name="download"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="download"
    inputVariable="Cid" outputVariable="SPECTResult"/>
</sequence>
...
```

(a)



(b)

Fig. 2. SPECT workflow code (a). Graphical representation of a SPECT workflow (b)

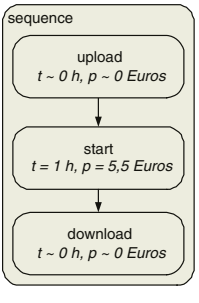
The `<qos-constraints>` element may contain several `<qos-constraint>` elements. Each `<qos-constraint>` element specifies one QoS constraint as a name/value pair. In the example above, the `<qos-constraints>` element of the `start` activity contains three `<qos-constraint>` elements. The `<qos-constraint>` element `beginTime` specifies the earliest possible start time of the reconstruction service, whereas the `<qos-constraint> endTime` specifies the latest possible completion time of the service. The `<qos-constraint> price` defines the maximal price for the service execution.

The attribute `ReqDesc` specifies the request descriptor of a service, which contains the performance relevant input data, e.g. number of iterations, size of the input data etc. The `wsla` attribute references the WSLA document of a VGE service. Besides QoS constraints, the WSLA document contains different conventions concerning the service execution, such as penalties to be paid if a service could not be executed within the agreed constraints.

In the example above, we only use QoS constraints for begin time, end time and price of the service execution. Additional QoS constraints could be added

```
...
<sequence name="SPECTSequence">
  <invoke name="upload"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="upload" inputVariable="uploadData"/>
  <invoke name="start"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="start" inputVariable="Cid">
    <qos-constraints
      ReqDesc="/home/gemss/workflow/ReqDescs/ReqDesc-1077647254348-12.xml"
      wsla="/home/gemss/workflow/wslas/wsla-107764727653-10.xml">
      <qos-constraint name="beginTime" value="2004-10-18T10:00:00.000+02:00"/>
      <qos-constraint name="endTime" value="2004-10-18T11:00:00.000+02:00"/>
      <qos-constraint name="price" value="5.50"/>
    </qos-constraints>
  </invoke>
  <invoke name="download"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="download" inputVariable="Cid"
    outputVariable="SPECTResult"/>
</sequence>
...
```

(a)



(b)

Fig. 3. SPECT workflow code with QoS constraints (a). Graphical representation (b)

by defining new `<qos-constraint>` elements. Note that in our example, QoS support is only defined for the reconstruction. Data staging activities, such as `upload` and `download`, are currently neglected because usually transfer times of image data are much shorter than reconstruction times.

4 QoS-Aware Workflow Engine

An appropriate Java-based negotiation and execution engine is being developed which (1) parses the QoS-aware workflow documents, (2) generates an intermediary representation of the workflow based on the workflow document and on the WSDL documents of the external services, (3) negotiates QoS parameters with services, (4) generates a Web Service for the whole workflow and deploys it, and finally (5) executes the workflow.

4.1 Architecture

Our QoS-aware execution engine consists of the following major components: the *workflow parser* and *unparser*, the *service generator*, the *QoS negotiator* and the *workflow executer*.

The *workflow parser* generates an intermediary representation of the workflow by parsing and validating the workflow document and WSDL document according to the schema definition.

The intermediary representation is a tree-based structure merging information of the according workflow document and WSDL documents of services. During the QoS negotiation user-specified QoS constraints are replaced by the actual constraints as offered by a service. After the QoS negotiation the *XML unparser* generates the resulting workflow document.

The *service generator* is responsible for the generation of a service which represents the whole workflow including the `wsdd` and other files necessary for service deployment. This process is performed on the fly by parsing and analyzing the workflow document and WSDL files. Services are deployed in a Tomcat servlet container [3] using Apache Axis [2]. The first prototype of the execution engine handles request/response and asynchronous invocations of the workflow by a client.

The *QoS negotiator* is defined as an abstract interface called `QoS handler` and is responsible for the negotiation of the QoS constraints of an activity. The `QoS handler` defines only the interface for the QoS negotiation and therefore different QoS negotiation strategies could be implemented, for example, auction-based negotiation strategies. The `QoS negotiator` automatically generates the `QoS Descriptor` with the constraints specified by the user and compares them with the QoS constraints offered by services.

The *workflow executer* is triggered by an external invocation of a workflow. The `Service Invocation Handler` is responsible for the invocation of services initiated by the workflow process, for the generation of SOAP requests and for the interpretation of responses.

4.2 Workflow Lifecycle

Figure 4 shows a lifecycle scenario for a QoS-aware workflow.

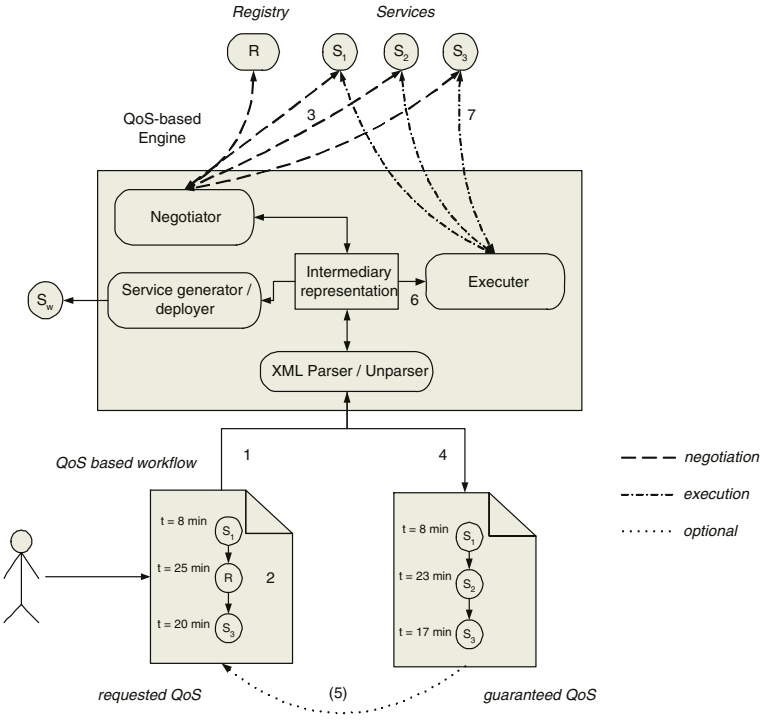


Fig. 4. QoS-aware workflow lifecycle

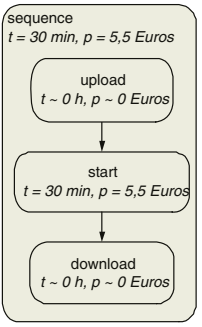
In the first step the user specifies a workflow document with requested QoS constraints (1). In most cases QoS negotiation only make sense if the user or the workflow execution engine may choose between several service providers. For this purpose the `wsdl` attribute of the invoke activity should specify a service registry (2) instead of a particular service endpoint. In this case a QoS negotiation (3) is performed with all services found in the registry and the service with the best offer is selected. If an activity specifies a particular service endpoint, the `QoS negotiator` contacts the service (3) and receives an offer. After the QoS negotiation the received QoS constraints are written to the original workflow document (4).

Figure 5 shows the medical image workflow after the QoS negotiation. The QoS constraints of the `start` activity are now those guaranteed by the selected VGE service.

As shown in Figure 5 we distinguish between QoS constraints of a basic activity, which reflect the QoS constraints of an external VGE service, and the

```
...
<sequence name="SPECTSequence">
  <qos-constraints
      ReqDesc="/home/gemss/workflow/ReqDescs/ReqDesc-107764727563-12.xml">
      <qos-constraint name="beginTime" value="2004-10-18T10:15:00.000+02:00"/>
      <qos-constraint name="endTime" value="2004-10-18T10:45:00.000+02:00"/>
      <qos-constraint name="price" value="5.5"/>
    </qos-constraints>
  <invoke name="upload"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="upload" inputVariable="uploadData"/>
  <invoke name="start"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="start" inputVariable="Cid">
    <qos-constraints
      ReqDesc="/home/gemss/workflow/ReqDescs/ReqDesc-107764727535-12.xml"
      wsla="/home/gemss/workflow/wslas/wsla-107764727653-10.xml">
      <qos-constraint name="beginTime" value="2004-10-18T10:15:00.000+02:00"/>
      <qos-constraint name="endTime" value="2004-10-18T10:45:00.000+02:00"/>
      <qos-constraint name="price" value="5.5"/>
    </qos-constraints>
  </invoke>
  <invoke name="download"
    wsdl="http://bridge.vcpc.univie.ac.at:9355/SPECT/appex?wsdl"
    portType="appex" operation="download" inputVariable="Cid"
    outputVariable="SPECTResult"/>
</sequence>
...
```

(a)



(b)

Fig. 5. SPECT workflow code with the agreed QoS constraints after negotiation (a). Corresponding graphical representation (b)

QoS constraints of a complex activity, which have to be calculated by the execution engine based on the QoS constraints of the underlying activities. The QoS constraints of the `start` activity represent the QoS constraints of a VGE service whereas the QoS constraints of the `SPECTSequence` are computed based on the QoS constraints of the nested activities.

If the user-specified QoS constraints cannot be met, the user may repeat the QoS negotiation (5) with new constraints or with new services and registries, respectively. If all constraints are satisfied, the execution of the workflow is scheduled according to the begin time of the first activity (6 and 7).

5 Related Work

BPEL allows the definition of graph-based and algebraic workflows considering the definition of communication channels in terms of partner links, advanced instance routing mechanisms, and fault handling. Presently, two BPEL execution engines exist: an IBM alphaworks execution engine BPWS4J [7] and a commercial product Collaxa [10]. The work presented in [17] at GGF10 proposes a Grid Process Execution Language (GPEL) and Grid-based workflow execution engine with the appropriate Grid-based monitoring and lifecycle support. The key issue of this approach is the representation of the current state of the workflow as an XML document enabling workflow monitoring and recovery. Work presented in [8] deals with theoretical concepts of a QoS-aware workflow defining QoS workflow metrics. A-GWL [11] focuses on the language constructs for an abstract

grid workflow language. Triana [14] is a component-based workflow language enabling the construction of visualized workflows by connecting predefined Triana units. An integrated BPEL reader allows the definition of BPEL workflows and invocation of external Web Services. FreeFluo [16] is a workflow execution engine developed by the MyGrid project [15]. The WSDL-based Web Service invocation works with two different languages. The first one is a subset of WSFL, the second one is a self-developed language named XScufl [22]. Freefluo supports the definition of in silico bioinformatics experiments and allows semantic description of workflows. However, none of the presented engines support QoS negotiation which is a crucial requirement for time critical applications.

6 Conclusion and Future Work

In this paper we presented QoS support for grid workflows addressing the special requirements of time-critical Grid applications. We defined extensions of BPEL for specifying QoS constraints for basic and complex activities. A prototype of QoS-aware workflow execution engine was presented that supports flexible service discovery based on QoS negotiation.

In the future we plan to explore dynamic QoS negotiation mechanisms where the QoS constraints of the particular activities can be modified during the workflow execution taking into account the actual state of the workflow execution. The development of a user-friendly GUI for specification and visualization of workflows is currently under way.

Acknowledgments

The work was partially supported by the Austrian Science Fund as part of the AURORA project under contract SFB F011-02 and by the European Union's GEMSS (Grid-Enabled Medical Simulation Services) Project under contract IST 2001-37153.

References

1. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana: *"Business Process Execution Language for Web Services Version 1.1"*, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ 2003
2. Apache AXIS http://ws.apache.org/axis/ 2004
3. Apache Tomcat 4.1.30 http://jakarta.apache.org/tomcat/ 2004
4. W. Backfrieder, M. Forster, S. Benkner and G. Engelbrecht: *Locally Variant VOR in Fully 3D SPECT within a Service Oriented Environment*, Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences, CSREA Press, Las Vegas, USA, 2003

5. S. Benkner, I. Brandic, G. Engelbrecht, R. Schmidt. *VGE - A Service-Oriented Environment for On-Demand Supercomputing*. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004
6. S. Benkner, G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S. E. Middleton, R. Schmidt: *GEMSS: Grid-infrastructure for Medical Service Provision*, HealthGrid 2004, Clermont-Ferrand, France, 2004
7. BPWS4J engine. http://www.alphaworks.ibm.com/tech/bpws4j 2004
8. J. Cardoso, A. Sheth, and J. Miller: *Workflow Quality of Service* , Enterprise Inter- and Intra-Organisational Integration - Building International Consensus, Kluwer Academic Publishers 2002
9. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, IBM: *Web Services Description Language (WSDL) 1.1*, http://www.w3.org/TR/wsdl 2001
10. Collaxa Engine http://www.collaxa.com 2004
11. T. Fahringer, S. Pllana, and A. Villazon. *A-GWL: Abstract Grid Workflow Language*. International Conference on Computational Science, Programming Paradigms for Grids and Metacomputing Systems. Krakow, Poland, 2004
12. The GEMSS Project: Grid-Enabled Medical Simulation Services, EU IST Project, IST-2001-37153, http://www.gemss.de/
13. F. Leyman: Web Service Flow Language WSFL 1.0, http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf IBM 2001
14. S. Majithia, M. S. Shields, I. J. Taylor, I. Wang: *Triana: A Graphical Web Service Composition and Execution Toolkit*. International Conference on Web Services, San Diego, USA, 2004
15. MyGrid Project http://www.mygrid.org.uk/ 2004
16. T. M. Oinn, M. Addis, J. Ferris, D. Marvin, R. M. Greenwood, C. A. Goble, A. Wipat, P. Li, T. Carver: *Delivering Web Service Coordination Capability to Users*. International World Wide Web Conference, NY, USA, 2004
17. A. Slominski: BPEL in Grids http://www.extreme.indiana.edu/swf-survey/GGF10_bpel_in_grids-2004-03-10.ppt 2004
18. SOAP Version 1.2. http://www.w3.org/TR/soap/
19. S. Thatte: XLANG http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
20. Web Service Level Agreement (WSLA) Language Specification. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, IBM 2001-2003
21. Web Service Security. SOAP Message Security 1.0, OASIS Standard 200401, March 2004
22. XScufl - http://www.ebi.ac.uk/∼tmo/mygrid/XScuflSpecification.html 2004

Transparent Fault Tolerance for Grid Applications

Paweł Garbacki[1], Bartosz Biskupski[2], and Henri Bal[3]

[1] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands
p.garbacki@ewi.tudelft.nl
[2] Department of Computer Science, Trinity College, Dublin, Ireland
biskupski@cs.tcd.ie
[3] Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
bal@cs.vu.nl

Abstract. A major challenge facing grid applications is the appropriate handling of failures. In this paper we address the problem of making parallel Java applications based on Remote Method Invocation (RMI) fault tolerant in a way transparent to the programmer. We use globally consistent checkpointing to avoid having to restart long-running computations from scratch after a system crash. The application's execution state can be captured at any time also when some of the application's threads are blocked waiting for the result of a (nested) remote method call. We modify only the program's bytecode which makes our solution independent from a particular Java Virtual Machine (JVM) implementation. The bytecode transformation algorithm performs a compile time analysis to reduce the number of modifications in the application's code which has a direct impact on the application's performance. The fault tolerance extensions encompass also the RMI components such as the RMI registry. Since essential data as checkpoints are replicated, our system is resilient to simultaneous failures of multiple machines. Experimental results show negligible performance overhead of our fault-tolerance extensions.

1 Introduction

Computational grids become increasingly important to solve computationally intensive and time consuming problems in science, industry, and engineering [3, 4, 21]. Since the failure probability increases with a rising number of components, fault tolerance is an essential characteristic of massively parallel systems. Such systems must provide redundancy and mechanisms to detect and localise errors as well as to reconfigure the system and to recover from error states.

Java's platform independence is well suited to a heterogeneous infrastructures that typify grid architectures. The object oriented nature of Java facilitates and code reuse significantly reduces development time. There is a wide variety of interfaces and language extensions that simplify parallel programming in Java not to mention Java threads and Remote Method Invocation (RMI) [2] which are

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 671–680, 2005.

parts of the Java specification [10]. Despite all these facilities, the Java Virtual Machine (JVM) [12] standard does not support fault tolerance.

In this paper we present the design, implementation and evaluation of a system that provides coordinated checkpointing and recovery for RMI-based parallel Java applications with a focus on grid computing. Our approach is novel in that it was designed to be transparent to the programmer, requiring minimal changes to the application code.

The contributions of this paper are the following: first we design and implement a Java bytecode transformer that makes the application programs resilient to failures by means of checkpoint-recovery. The comprehensive compile time analysis significantly reduces the overhead incurred by the application code modifications. Second, we present a technique that allows checkpointing even inside (nested) remote method calls. Third, we provide mechanisms which enable RMI components to recover from a system crash in a completely transparent manner. Finally, we present performance results for classic parallel applications. As a yardstick, we compare the performance of Java applications with and without the fault tolerance extensions.

The rest of this paper is organized as follows. Section 2 lists the related work. Sections 3 and 4 describe the capturing and reestablishing of the state of a single process and the whole application, respectively. Section 5 introduces the subsystem responsible for handling checkpoints initiated inside remote method calls. Section 6 describes fault tolerant RMI. Section 7 presents the performance of our system. Finally, Section 8 concludes this paper.

2 Related Work

The importance of fault tolerance in grid computing has already been recognised by the establishment of the Grid Checkpoint Recovery Working Group [16]. Its purpose is to define user-level mechanisms and grid services for fault tolerance.

We are not aware of any other project that is specifically addressing the provision of transparent fault tolerance support for grid applications written in Java. There is, however, a considerable amount of work in various related areas. The problem of saving the complete state of a Java program was investigated in the context of mobile agents migration. The Nomads [17] and Sirac [5] projects modify the JVM to capture the execution state of the application, which makes them inappropriate for the heterogeneous grid environment where different machines may have different JVMs installed. The CIA [11] mobile agents platform uses Java Platform Debugger Architecture (JPDA) API [1] which is not compatible with most Just In Time (JIT) compilers and thus causes increased execution overhead. The last category includes projects that insert additional instructions to the application code. This is the case for the Brakes [8], JavaGo [15], and Wasp [9] projects.

3 Capturing and Reestablishing Local State

A grid application is composed of a set of (multithreaded) processes distributed over a number of nodes, usually on one process per node basis. A *local checkpoint* is then a snapshot of the local state of a process. A *global checkpoint* is a set of all local checkpoints saved on nonvolatile storage to survive system failures.

The process execution state consists of the Java stacks of all local threads. Each Java stack consists of frames that are associated with method invocations. A new frame is pushed onto the stack each time a method is invoked and popped from the stack when the method returns. A frame includes the local variables, the operand stack with partial results of the computations, and the program counter which indicates the next instruction to be executed.

A thread has access only to data stored in the currently executing method frame due to the strict Java security policies, and therefore there is no straightforward way to preserve the whole Java stack. In our system we use an approach similar to that proposed by the Brakes [8] project. The state capturing in Brakes is initiated explicitly by placing a checkpoint function call in the application source code. Brakes provides a post-compiler which instruments the application bytecode by inserting additional code blocks that do the actual capturing and reestablishing of the execution state. The Brakes bytecode transformer inserts a code block after every method invocation, which saves in the checkpoint the stack frame of the current method and returns control to the previous method on the stack. This process continues until the whole Java stack is saved. The process of reestablishing a thread's state is similar but restores the stack frames in reverse order. The bytecode transformer inserts code at the beginning of each method, which restores the stack frame of the current method and invokes the next method whose state is saved in the checkpoint, consequently restoring the next stack frame. The process continues until the whole stack is rebuilt.

A significant improvement that we made to the original Brakes algorithm is that we added the analysis of the methods call graph in order to detect and modify only these methods and method invocations that could lead to the execution state capturing. This modification has been proven (see Sect. 7) to dramatically decrease the number of rewritten methods and thus reduce the overhead caused by transforming the application. The methods call analyser finds a set of all methods in all user classes whose invocation can lead to state capturing. Initially, the set contains just one method — the internal method that directly initiates state capturing. In each iteration the algorithm adds to the set new methods that can invoke methods already in the set. The algorithm stops when no methods were added in the last iteration.

4 Capturing and Reestablishing Global State

The checkpointing mechanisms described in Sect. 3 apply only to threads running within the same address space. We extend the applicability of our checkpointing mechanisms to the class of distributed applications by using the *coordinated*

checkpointing scheme [18]. As its name suggests, in coordinated checkpointing all the threads running on different nodes have to synchronise before writing their states to stable storage. Global coordination guarantees that the saved state is automatically *globally consistent* [6].

The global thread coordination is performed in two phases. In the first phase threads are synchronised locally at each node. The goal of the second phase is to synchronise all nodes. The coordination of threads running on the same node is performed with the help of the *local coordinator*, a component deployed on each node. When a thread is ready for a global checkpoint it notifies its local coordinator. Once all local coordinators receive confirmation from all threads the distributed phase of the global coordination process begins. The distributed synchronisation algorithm proposed by us is based on software trees [19]. A tree-based barrier is known to provide excellent performance and to scale well to large machines [14]. Although this method was originally designed for multiprocessors, it can easily be adapted to a distributed environment.

In order to make our system not only resilient to software faults but also to hardware faults, we replicate the checkpointed data among different machines. This way even in a situation when a stable storage device on one of the nodes crashes, the checkpoint can be still retrieved from a remote location.

When a failure occurs, the processes that were running on the crashed nodes are restored from the latest checkpoint on the backup nodes. Each of the backup nodes runs a simple service capable of obtaining the local checkpoint of a crashed process and initiating its recovery (described in Sect. 3).

There are several situations in which the failure may be detected. First, the user application can explicitly invoke the scanning procedure that will check which nodes are down. Second, the failure may be detected during the global barrier synchronisation. Third, a crash of a remote object may be discovered by the fault-tolerant RMI described in Sect. 6. Finally, there is a dedicated thread running on every node that checks periodically whether all remote processes are up and running.

5 Capturing and Reestablishing Distributed Thread's State

To increase the level of programming transparency, we allow the programmer to initiate the state capturing at any stage of the program execution, also when some of the threads perform remote method calls. To our knowledge our approach is the first to manage this problem in a completely distributed way, without any central components. Before presenting our solution, we first explain why saving and restoring of a state of a thread performing a remote method call is challenging, and also introduce some terminology.

Since Java threads are bound to the virtual machine in which they were created, a remote method execution is mapped to two different Java threads: the *client thread* that initiated the call, and the *server thread* that executes the remote method on the remote node. These two threads are both representatives

for the same *distributed thread of control* [22, 7]. The checkpoint initiated inside a remote method call should contain the state of both server and client threads. In a local execution environment, the JVM thread identifier offers a unique reference for a single computation entity. Java, however, does not offer any interfaces that allow us to recognise two threads as parts of the same distributed thread. To cope with this problem we extend Java programs with the notion of *distributed thread identity* [22]. Propagation of globally unique identifiers allows for the identification of local computations as parts of the same distributed computation entity.

Using the introduced terminology we describe the idea of a distributed thread state capturing. Each local thread in our system has an associated identity of the distributed computation of which it is part. The identity is generated when the distributed computation is started, that is, when the oldest local thread which is part of the distributed computation is instantiated. The remote thread identity is sent along with the remote method call. It is done by extending the signature of the remote method with a parameter representing the distributed thread identity. Now suppose that the checkpoint was requested inside a remote method call. We start from capturing the state of the server thread which initiated the checkpoint. The context object containing the serialized state of the server thread is stored on the server machine under the key representing the distributed thread identity. After its state was captured, the server thread throws a special type of exception notifying the client thread that the checkpoint was requested. This procedure is repeated until the contexts of all local threads have been saved. The distributed thread state reestablishing is a reverse process.

6 Fault Tolerant RMI

A strong point of Java as a language for grid computing is the integrated support for parallel and distributed programming. Java provides Remote Method Invocation (RMI) [2] for transparent and efficient [13, 20] communication between JVMs. However, it does not have any support for fault tolerance. Therefore, we developed mechanisms that provide fault tolerance for the Java RMI components. We provide a replicated *RefStore* server that maintains remote objects references, mechanisms that allow remote objects and their stubs to transparently recover from a system crash, and a fault tolerant RMI registry.

The replicated RefStore server was developed for the purpose of storing remote references to recovered fault tolerant remote objects. The remote reference is a standard Java object (a component of every stub) containing the address of the hosting node, a communication port number, and a key that together uniquely identify the remote object. When a fault tolerant remote object recovers from a crash, it automatically registers its new remote reference in the RefStore server. When a stub cannot connect to the object using its old remote reference, it retrieves the new remote reference from the RefStore.

In order to release the programmer from the burden of detecting failures and updating remote references, a transformation algorithm that analyses a user's

stub classes and automatically generates exception handlers was developed. The exception handler is invoked when the stub cannot connect to the fault tolerant remote object. When it happens, the recovery process is initiated. After successful recovery, the stub automatically retrieves the new remote reference from the RefStore server and reconnects to the remote object using its new location.

The Java RMI registry [2] is typically used for exchanging stubs for remote objects. However, since the node on which the registry is running may also fail, we provide an implementation of a fault-tolerant RMI registry service that is checkpointed together with the whole application.

To summarize, each component of the fault-tolerant RMI system is either replicated or checkpointed, and so there is no single point of failure. Moreover, since no modifications in the application code are needed, our fault-tolerant RMI is completely transparent to the programmer.

7 Performance Evaluation

In this section we study the impact of our fault-tolerance extensions on the performance of the distributed applications. All tests were performed on the DAS2 cluster[1] of 1GHz Pentium III processors, running Linux, and connected by a Myrinet network.

We investigate the overhead incurred by the checkpointing extensions on two applications, namely Successive Over Relaxation (SOR) and Traveling Salesman Problem (TSP). These applications were selected as being challenging for the checkpointing system. Complicated control flow scheme and non-negligible amounts of temporary data pose difficulties for making these applications fault tolerant manually.

SOR is an iterative method for solving discretised Laplace equations on a grid. The program distributes the grid row-wise among the processors. Each processor exchanges its row of the matrix with its neighbors at the beginning of each iteration.

TSP finds the shortest route among a number of cities using a parallel branch-and-bound algorithm, which prunes large parts of the search space by ignoring partial routes already longer than the current best solution. We divide the whole search tree into many small ones to form a job queue. Every worker thread will get jobs from this queue until the queue is empty.

We measure the performance overhead during the normal execution (without initiating the state capturing) introduced in these applications by the fault-tolerance extensions. This overhead is generated by additional conditional statements placed after method calls which may initiate state capturing, and by replacing the standard Java RMI with its fault-tolerant counterpart. We argue that the overhead caused by the extra conditional statements is negligible since they are sensibly placed and thus rarely invoked.

[1] A detailed description of the DAS2 cluster architecture can be found at *http://www.cs.vu.nl/das2*

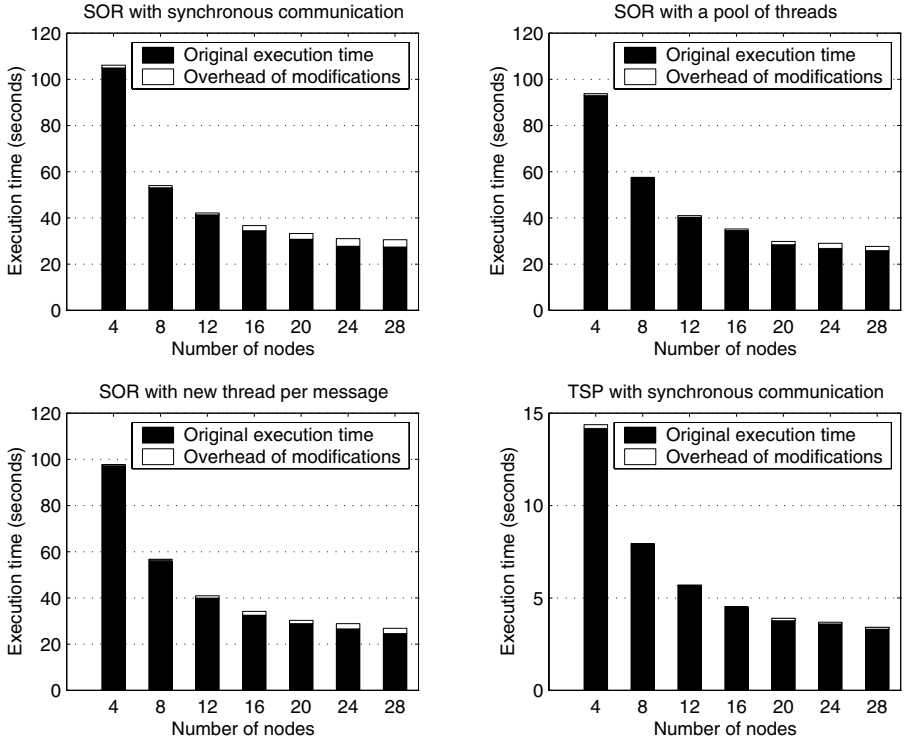


Fig. 1. The performance overhead incurred by our fault-tolerance extensions

We first asses the performance when no checkpoints are taken. Figure 1 presents the comparison of the execution times of the original and the transformed applications in relation to the number of nodes used for the computation in that case. Three variants of the SOR algorithm (for a matrix 1000x10000 and 200 iterations) and one variant of TSP (for 17 cities) were used during the measurements. The first version of the SOR algorithm uses synchronous communication. The second version is based on asynchronous communication scheme with a pool of threads which are reused for sending messages. The last variant of the SOR algorithm starts a new thread for every data send request. The TSP application uses synchronous communication.

The measurements show that overhead in SOR incurred by the fault-tolerance extensions increases with the number of nodes. The overhead caused by the stubs transformations is proportional to the number of remote method invocations which is higher for larger number of nodes. The highest performance overhead of 9% was observed for the SOR application with synchronous communication. In the case of synchronous communication, the overhead incurred by the stubs transformations affects directly the computation thread, thus slowing down the whole application. The performance degradation of the asynchronous versions

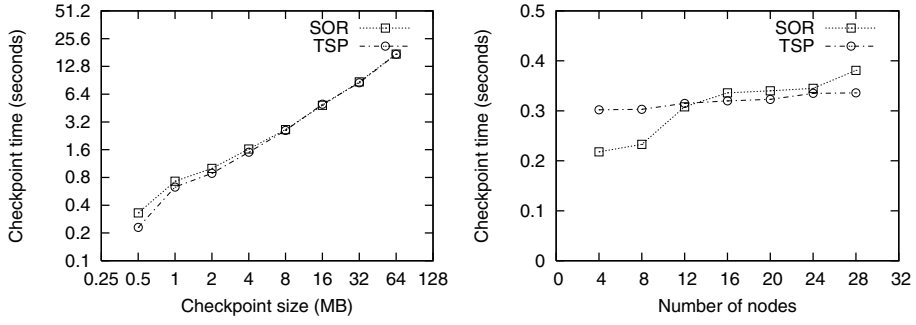


Fig. 2. The duration of the checkpointing process as a function of the checkpoint size (left) and the number of nodes (right). Note the logarithmic scale on both axes of the left plot

of SOR affects directly only the communication threads that are running in the background. The highest observed performance losses in the asynchronous versions with a thread pool and a new thread per message were 5% and 6%, respectively. A slightly higher overhead in the latter case comes from the fact that creating a new thread for each message requires registering this thread in our system. The bottom-right plot of Fig. 1 presents the results of the TSP application. The fault-tolerance extensions in this application hardly influence its performance. The highest observed overhead was less than 2%. Since processes do not communicate with each other as frequently as in the SOR algorithm, the overhead caused by the fault-tolerant RMI is much lower.

We now turn to the time that is needed to take a distributed checkpoint (Fig. 2). Clearly, capturing the state of a process that contains more data takes longer. Similarly the number of nodes in the system is not without influence on the overall performance of the checkpoint. The delay introduced by the global barrier synchronisations grows with the number of nodes involved.

The left half of Fig. 2 shows how the size of the checkpointed data relates to the time needed to take a globally consistent checkpoint of the applications running on 20 nodes (note the logarithmic scale on both axes). For the distributed checkpoint performance evaluation we used the most complex variant of the SOR algorithm, namely the asynchronous communication with the thread pool. As one could expect, the time of a checkpoint can be approximated by a linear function of the data size. The linear dependency on this logarithmic plot is however much weaker for smaller (less than 3MB) than for bigger checkpoints. For smaller checkpoints the state capturing time is dominated by the efficiency of the state saving extensions. Checkpoints that contain more data move the overhead from the data serialisation to the data replication phase, which is much more data size dependent.

The plot presented in the right half of Fig. 2 shows the influence of the number of nodes on the performance of the checkpoint. The size of the checkpointed

data was approximately the same for all applications — 500KB. The overhead of the checkpointing phase is determined by two factors. The performance of the global barrier synchronisation algorithm depends on the number of nodes involved. Furthermore, different threads need different amount of time to capture their states. The variations of the state serialization times among different threads accumulate resulting in considerable delays. The results of the experiments show however that our system can deal with a higher number of nodes without excessive performance loss.

As we described in Sect. 3, we optimised the original Brakes algorithm by rewriting only those methods that may lead to the checkpoint request. We measured the performance gain due to this optimization for the SOR and TSP applications running on 8 nodes. The number of rewritten methods was reduced for SOR from 53 to 5 and for TSP from 37 to 2. This resulted in a performance gain of over 10% in the case of SOR and over 30% in the case of TSP.

8 Conclusions

In this paper we have presented a complete solution for making regular grid applications written in Java fault tolerant. The high level of programming transparency and independence from a particular JVM enable easy integration with existing applications and grid services. The experiments show that our approach has a very low performance overhead during normal program execution, and scales well with the system size.

Acknowledgments

The authors would like to thank Jim Dowling, Dick Epema, Thilo Kielmann, and Tim Walsh for their valuable comments on the draft version of this paper. This work was partly supported by the DBE Project, IST 6th Framework Programme.

References

[1] Java platform debugger architecture (jpda). `http://java.sun.com/products/jpda/`.

[2] Java remote method invocation specification. revision 1.10, jdk 1.5.0, 2004. `http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf`.

[3] G. Allen, W. Benger, T. Goodale, H. Ch. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf. The cactus code: A problem solving environment for the grid. In *The Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC9)*, Pittsburgh, PA, USA, August 2000.

[4] D. C. Arnold and J. Dongarra. The netsolve environment: Progressing towards the seamless grid. In *International Workshop on Parallel Processing*, Toronto, Canada, August 2000.

[5] S. Bouchenak. Making java applications mobile or persistent. In *Conference on Object-Oriented Technologies and Systems*, San Antonio, TX, USA, January 2001.

[6] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[7] R. Clark, E. Jensen, and F. Reynolds. An architectural overview of the alpha real-time distributed kernel. In *USENIX Winter Conference*, San Diego, CA, USA, January 1993.

[8] T. Coninx, E. Truyen, B. Vanhaute, Y. Berbers, W. Joosen, and P. Verbaeten. On the use of threads in mobile object systems. In *6th ECOOP Workshop on Mobile Object Systems*, Sophia Antipolis, France, June 2000.

[9] S. Fuenfrocken. Transparent migration of java-based mobile agents. In *Second International Workshop on Mobile Agents*, Stuttgart, Germany, September 1998.

[10] J. Gosling, B. Joy, G. L. Steele Jr., and G. Bracha. *The Java Language Specification*. Addison Wesley, second edition, 2000. `http://java.sun.com/docs/books/jls/`.

[11] T. Illman, T. Krueger, F. Kargl, and M. Weber. Transparent migration of mobile agents using the java platform debugger architecture. In *The Fifth IEEE International Conference on Mobile Agents*, Atlanta, GA, USA, December 2001.

[12] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison Wesley, second edition, 1999. `http://java.sun.com/docs/books/vmspec/`.

[13] J. Maassen, R. van Nieuwpoort, R. Veldema, H. Bal, T. Kielmann, C. Jacobs, and R. Hofman. Efficient java rmi for parallel programming. *ACM Transactions on Programming Languages and Systems*, 23(6):747–775, November 2001.

[14] J. M. Mellor-Crummey and M. L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, February 1991.

[15] T. Sekiguchi, H. Masuhara, and A. Yonezawa. A simple extension of java language for controllable transparent migration and its portable implementation. In *3rd International Conference on Coordination Models and Languages*, Amsterdam, The Netherlands, April 1999.

[16] Nathan Stone, Derek Simmel, and Thilo Kielmann. GWD-I: An architecture for grid checkpoint recovery services and a GridCPR API. Grid Checkpoint Recovery Working Group Draft 3.0, Global Grid Forum, `http://gridcpr.psc.edu/GGF/docs/draft-ggf-gridcpr-Architecture-2.0.pdf`, May 2004.

[17] N. Suri, J. Bradshaw, M. Breedy, P. Groth, A. G. Hill, and R. Jeffers. Strong mobility and fine-grained resource control in nomads. In *Agent Systems and Applications / Mobile Agents*, Zurich, Switzerland, September 2000.

[18] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[19] P. Tang and P. C. Yew. Algorithms for distributing hot spot addressing. Technical report, Center for Supercomputing Research and Development, University of Illinois Urbana-Champaign, January 1987.

[20] R. V. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H. E. Bal. Ibis: An efficient java-based grid programming environment. In *Joint ACM Java Grande - ISCOPE 2002 Conference*, Seattle, WA, USA, November 2002.

[21] R. V. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H. E. Bal. Satin: Simple and efficient java-based grid programming. In *AGridM 2003 Workshop on Adaptive Grid Middleware*, New Orleans, LA, USA, September 2003.

[22] D. Weyns, E. Truyen, and P. Verbaeten. Distributed threads in java. In *International Symposium on Parallel and Distributed Computing*, Iasi, Romania, July 2002.

Learning Automata Based Algorithms for Mapping of a Class of Independent Tasks over Highly Heterogeneous Grids

S. Ghanbari and M.R. Meybodi

Soft Computing Laboratory,
Computer Engineering Department and Information Technology,
Amirkabir University, Tehran Iran
saeed_ghanbari@yahoo.com , meybodi@ce.aut.ac.ir

Abstract. Computational grid provides a platform for exploiting various computational resources over wide area networks. One of the concerns in implementing computational grid environment is how to effectively map tasks onto resources in order to gain high utilization in the highly heterogeneous environment of the grid. In this paper, three algorithms for task mapping based on learning automata are introduced. To show the effectiveness of the proposed algorithms, computer simulations have been conducted. The results of experiments show that the proposed algorithms outperform two best existing mapping algorithms when the heterogeneity of the environment is very high.

1 Introduction

Owing to advances in computational infrastructure and networking technology, construction of large-scale high performance distributed computing environment, known as *computational grid*, is now possible. Computational grid enables the sharing, selection, and aggregation of geographically distributed heterogeneous resources for solving large scale problems in science, engineering and commerce. Numerous efforts have been exerted focusing on various aspects of grid computing including resource specifications, information services, allocation, and security issues. A critical issue to meeting the computational requirements on the grid is the scheduling.

Ensuring a favorable efficiency over computational grid is not a straightforward task, where a number of issues make scheduling challenging even for highly parallel applications. Resources on the grid are typically shared and undedicated so that the contention made by various applications results in dynamically fluctuating delays, capricious quality of services, and unpredictable behavior, which further complicate the scheduling. Regarding to these hurdles, the scheduling of applications on computational grids have become a major concern of multitude efforts in recent years[9].

In mixed-machine *heterogeneous computing* (HC) environments like computational grids, based on application model characterization, platform model

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 681 – 690, 2005.

characterization and mapping strategy characterization, there are various definitions for scheduling[6]. Ideal sorts of applications for computational grid are those composed of independent tasks, which tasks can be executed in any order and there is no inter-task communication (i.e. totally parallel) [1][12]. There are many applications of such feature including data mining, massive searches (such as key breaking), parameter sweeps, Monte Carlo simulations[2], fractals calculations (such as Mandelbrot), and image manipulation applications (such as tomographic reconstruction[3]). Computational grid platform model consists of different high-performance machines, interconnected with high-speed links. Each machine executes a single task at a time (i.e. no multitasking) in the order to which the tasks are assigned. The matching of tasks to machines and scheduling the execution order of these tasks is referred to as mapping. The general problem of optimally mapping tasks to machines in an HC suite has been shown to be NP-complete [11].

In this paper, we present three algorithms based on learning automata for mapping metatask over HC. Through computer simulations we show that the proposed algorithms outperform the best existing mapping algorithms when the heterogeneity of the environment is very high.

This paper is organized as follows: Section 2 discusses the related works. Section 3 introduces learning automata. Section 4 explains the model of the Grid and the definitions used in later sections. Section 5 introduces the proposed learning automata based algorithms. In Section 6, experimental results are discussed, and section 7 provides the conclusion.

2 Related Works

Existing mapping algorithms can be categorized into two classes[4]: on-line mode (immediate) and batch mode. In on-line mode, a task is mapped onto a host as soon as it arrives at the scheduler. In the batch mode, tasks are collected into a set that is examined for mapping at certain intervals called *mapping events*. The independent set of tasks that is considered for mapping at the mapping events is called a *metatask*. The on-line mode is suitable for low arrival rate, while batch-mode algorithms can yield higher performance when the arrival rate of tasks is high because there are a sufficient number of tasks to keep hosts busy between the mapping events, and scheduling is done according to the resource requirement information of all tasks in the set[4]. The objective of most mapping algorithms is to minimize *makespan*, where makespan is the time needed for completing the execution of a metatask. Minimizing *makespan* yields to higher throughput.

Reported batch mode heuristics are Min-Min, Max-Min, Genetic Algorithm (GA), Simulated Annealing (SA), Genetic Simulated Annealing (GSA), A^* search, Suffrage[5][4], and *Relative Cost* (RC) [7] . Experimental results show that among batch-mode heuristics, Min-Min and GA give lower *makespan* than other heuristics[5], and RC further outperforms both GA and Min-Min[7].

RC introduces two essential criteria for a high-quality mapping algorithm for heterogeneous computing systems: *matching* which is to better match the tasks and machines, and *load balancing* which is to better utilize the machines. It is shown that in order to minimize the makespan, matching and system utilization should be

maximized[7]. However, these design goals are in conflict with each other because mapping tasks to their first choice of machines may cause load imbalance. Therefore, the mapping problem is essentially a tradeoff between the two criteria. Two out of three proposed algorithms in this paper resolve mapping by optimizing matching and load balancing.

3 Learning Automata

Learning Automata are adaptive decision-making devices operating on unknown random environments. A Learning Automaton has a finite set of actions and each action has a certain probability (unknown to the automaton) of getting rewarded by the environment of the automaton. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction on the system. If the learning algorithm is chosen properly, then the iterative process of interacting on the environment can be made to result in selection of the optimal action. Learning Automata can be classified into two main families: fixed structure learning automata and variable structure learning automata (VSLA) [8]. In the following, the variable structure learning automata which will be used in this paper is described.

A VSLA is a quintuple $< \alpha, \beta, p, T(\alpha, \beta, p) >$, where α, β, and p are an action set with r actions, an environment response set, and the probability set p containing r probabilities, each being the probability of performing every action in the current internal automaton state, respectively. The function of T is the reinforcement algorithm, which modifies the action probability vector p with respect to the performed action and received response. If the response of the environment takes binary values learning automata model is P-model and if it takes finite output set with more than two elements that take values in the interval [0,1], such a model is referred to as Q-model, and when the output of the environment is a continuous variable in the interval [0,1], it is refer to as S-model. Assuming $\beta \epsilon [0,1]$, a general linear schema for updating action probabilities can be represented as follows. Let action i be performed then:

$$p_j(n+1) = p_j(n) + \beta(n)[b/(r-1) - bp_j(n)] - [1-\beta(n)]ap_j(n) \quad \forall j \quad j \neq i \tag{1}$$

$$p_i(n+1) = p_i(n) - \beta(n)bp_i(n) + [1-\beta(n)]a[1-p_i(n)] \tag{2}$$

where a and b are reward and penalty parameters. When $a=b$, the automaton is called L_{RP}. If $b=0$ the automaton is called L_{RI} and if $0<b<<a<1$, the automaton is called $L_{R\varepsilon P}$. For more Information about learning automata the reader may refer to [8].

4 Simulation Model

This section presents a general model of the computational grid. The environment consists of the heterogeneous suite of machines which will be used to execute the application. The scheduling system consists of the automata, and the model of the application and the HC suite of machines.

The application and HC suite of machines are modeled as the estimate of the expected execution time for each task on each machine, which is known prior to the execution and contained within a $\tau \times \mu$ *ETC* (Expected Time to Compute) *matrix,* where τ is the number of tasks and μ is the number of machines. One row of the *ETC* matrix contains the estimated execution times for a given task on each machine. Similarly, one column of the *ETC* matrix consists of the estimated execution times of a given machine for each task in the metatask. Thus, for an arbitrary task s_i and an arbitrary machine m_j, $ETC(s_i,m_j)$ is the estimated execution time of s_i on m_j.

We define $\psi^{(n)}(i)=j$ as a general mapping from the task domain $i=1,\ldots,\tau$ to the machine domain $j=1,\ldots,\mu$ at iteration n. The load of each machine, which is denoted by $\theta^{(n)}(j)$, is defined as the time taken to execute all the assigned tasks:

$$\theta^{(n)}(j) = \sum ETC(k, j),\ j = \psi^{(n)}(k)\ 1 \le k \le \tau \tag{3}$$

The maximum of $\theta^{(n)}(j)$, over $1 \le j \le \mu$, is the metatask execution time, which is referred to as *makespan*, denoted by $T_{\mu}^{(n)}$.

5 Proposed Learning Automata Model

The learning automata model is constructed by associating every task s_i in the metatask with a variable structure learning automaton, which is represented by a 3-tuple$(a(i),\beta(i),A(i))$. Each action of an automaton is associated with a machine, and since the tasks can be assigned to any of the μ machines, the action set of all learning automata are identical. Therefore, for any task s_i, $1 \le i \le \tau$, $a(i)=m_1,m_2,\ldots,m_\mu$($m_i$ is the i^{th} machine), and $\beta(i) \epsilon [0,1]$, where $\beta(i)$ closer to 0 indicates that the action taken by the automaton of task s_i is favorable to the system, and closer to 1 indicates an unfavorable response. Reinforcement scheme used to update action probabilities of learning automata is L_{RI}.

To determine the goodness of an action taken by an automaton, we propose three different algorithms. The first algorithm calculates $\beta(i)$ for each automaton $A(i)$ according to the reduction made in *makespan* and the load of the selected machine. The second and third algorithms calculate the goodness of an action based on improvement made in matching and load balancing.

5.1 Algorithm No.1

The algorithm No.1 (A1) determines the $\beta^{(n)}(i)$ at iteration n for each automaton $A(i)$ by considering *makespan* and load of the chosen machine. Algorithm A1 interprets the environment as P-model; therefore $\beta^{(n)}(i) \epsilon \{0,1\}$. Makespan at iteration n may be greater, less than, or equal to makespan at iteration $n-1$. Similarly, load of the machine chosen by automaton $A(i)$ at iteration n may be greater, less than, or equal to load of the machine chosen by the automaton at iteration $n-1$. Therefore, regarding to makespan and the load of the chosen machine in two consecutive iterations, nine states are possible. To determine the $\beta^{(n)}(i)$, we associate a probability value to each nine possible state, which determines the probability of rewarding the chosen action. Probability one means that the chosen action will be rewarded. Table 1 shows the values, where D, U and I stand for decrease, remaining unchanged, and increase, respectively.

Table 1. Reward probability associated with each state

Makespan	*Load of chosen machine*	*Rewarding probability*
D	*D*	1
D	*U*	0.875
D	*I*	0.75
U	*D*	0.625
U	*U*	0.5
U	*I*	0.375
I	*D*	0.25
I	*U*	0.125
I	*I*	0

Algorithm Al is suitable for situations that the information used to evaluate the environment response is the load of machines.

5.2 Algorithm No.2

As mentioned in section 2, it is shown that to minimize the *makespan*, matching and system utilization must be maximized. Algorithm No.2 (A2) evaluates the response to the learning automata by considering these two criteria. Matching of tasks and machines can be measured by a parameter, *matching proximity*, which is defined as follows:

$$\eta = \frac{\sum_{1 \le i \le \tau} ETC(i, \psi_{\min}(i))}{\sum_{1 \le i \le \tau} ETC(i, \psi(i))} \tag{4}$$

where $\eta \le 1$, and $\psi_{min}(i)$ is the ideal matching. Ideal matching is defined as executing every task on the machine with the shortest execution time. It is defined as follows:

$$\psi_{\min}(i) = j \text{ such that } ETC(i, j) = \min_{1 \le q \le \mu} ETC(i, q) \tag{5}$$

when $\eta = 1$, we have the ideal matching. *System utilization* is defined as follows:

$$\delta = \frac{\sum_{1 \le j \le \mu} \theta(j)}{\mu \times T_{\mu}} \tag{6}$$

When the system is completely balanced, $\delta = 1$; otherwise $\delta < 1$.

Algorithm A2 reduces the mapping problem to an optimization problem with matching proximity and system utilization as objective functions. Algorithm A2 interprets the environment as S-model; therefore, $\beta^{(n)}(i)$ is in *[0,1]*.

To evaluate the contribution of each automaton to the improvement of matching and system utilization, we define two parameters, *partial contribution to matching*(PCM), and *partial contribution to load balancing*(PCL). Input to each automaton is a linear combination of PCM (denoted by $\eta^{(n)}(i)$), and PCL (denoted by $\delta^{(n)}(i)$):

$$\beta^{(n)}(i) = \eta^{(n)}(i)\lambda_{\eta} + \delta^{(n)}(i)\lambda_{\delta} \text{ where } \lambda_{\eta} + \lambda_{\delta} = 1 \tag{7}$$

λ_η and λ_δ are weights associated with PCM and PCL, respectively. PCM for each automaton *A(i)* at iteration *n* is evaluated as:

$$\eta^{(n)}(i) = \frac{ETC(i,\psi^{(n)}(i)) - ETC(i,\psi_{\min}(i))}{ETC(i,\psi_{\max}(i)) - ETC(i,\psi_{\min}(i))} \tag{8}$$

where $\psi_{max}(i)$ is the *worst matching* which is defined as mapping each task to a machine with the longest execution time; it is defined below

$$\psi_{\max}(i)=j \text{ such that } ETC(i,j) = \max_{1\le q\le \mu} ETC(i,q) \tag{9}$$

The closer $\eta^{(n)}(i)$ to 0, the more favorable the response from the environment as far as the matching is concerned. In the case that the automaton selects the machine with the worst matching, $\eta^{(n)}(i)$ is evaluated to 1.

PCL for each automaton *A(i)* at iteration *n* is evaluated as:

$$\partial^{(n)}(i) = \frac{\theta^{(n)}(\psi^{(n)}(i))}{T_{\mu}^{(n)}}(1 - e^{-\frac{1}{2}(\frac{\delta^{(n)}-1}{0.1})^2}) \tag{10}$$

The former part of the above expression is close to 0 when the chosen machine has a load less than the maximum load. In this way, the learning automata are encouraged to choose machines with low loads, thus, they are guided in a way to decrease the distance between the maximum load and the minimum load. The latter part of the expression is a Gaussian function, which gets closer to 0 as the system utilization increases; therefore, when the load is relatively balanced, PCL of each automaton is close to 0. Unlike algorithm A1, algorithm A2 requires information about the estimation of execution time of each task on each machine.

5.3 Algorithm No.3

Algorithm No.3 (A3) interprets the environment as a Q-Model environment. Like algorithm A2, it uses matching proximity and system utilization as objective functions. PCL and PCM are evaluated in the same way as algorithm A2, and used to produce the environment response. But, in algorithm A3, PCL and PCM are interpreted as probabilities, where PCL determines the probability that the learning automaton receives unfavorable response as far as system utilization is concerned, and PCM determines the probability that the learning automaton receives unfavorable response as far as matching is concerned. The environment response is evaluated as below:

$$\beta^{(n)}(i) = I(\eta^{(n)}(i))\lambda_\eta + I(\delta^{(n)}(i))\lambda_\delta \text{ where } \lambda_\eta + \lambda_\delta = 1 \tag{11}$$

λ_η and λ_δ are the weights associated with PCM and PCL, respectively. *I(p)* is an indicator function which returns 1 with the probability of *p*, and 0 with the probability of *1-p*. Therefore, the input to each automaton $\beta(i)$ is in $\{0,\lambda_\eta,\lambda_\delta,1\}$. In contrast to algorithm A2, algorithm A3 evaluates environment response stochastically, which allows the learning automata to jump local minimums in their search space.

6 Experiments

In this section the proposed algorithms are tested and compared with algorithms Min-Min and RC because these two algorithms are the best existing algorithms. For the simulation studies, *ETC* matrices were generated using the method presented in [4]. Initially, a $\tau \times 1$ baseline column vector, B, of floating point values is created. Let ω_b be the upper bound of the range of possible values within the baseline vector. The baseline column vector is generated by repeatedly selecting a uniform random number, $x_b^i \epsilon [1, \omega_b)$, and letting $B(i)=x_b^i$ for $1 \leq i \leq \tau$. Next, the rows of the *ETC* matrix are constructed. Each element $ETC(s_i,m_j)$ in row i of the *ETC* matrix is created by taking the baseline value, $B(i)$, and multiplying it by a uniform random number, $x_r^{i,j}$, which has an upper bound of ω_r. This new random number, $x_r^{i,j} \epsilon [1, \omega_r)$, is called a row multiplier. One row requires μ different row multipliers, $1 \leq j \leq \mu$. Each row i of the ETC matrix can then be described as $ETC(s_i,m_j) = B(i) \times x_r^{i,j}$, for $1 \leq j \leq \mu$. (The baseline column itself does not appear in the final ETC matrix.) This process is repeated for each row until the $\tau \times \mu$ *ETC* matrix is full. Therefore, any given value in the ETC matrix is within the range $[1, \omega_b \times \omega_r)$.

The amount of variance among the execution times of tasks in the metatask for a given machine is defined as task heterogeneity. Task heterogeneity is varied by changing the upper bound of the random numbers within the baseline column vector. High task heterogeneity was represented by ω_b=3000 and low task heterogeneity used ω_b=100. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Machine heterogeneity was varied by changing the upper bound of the random numbers used to multiply the baseline values. High machine heterogeneity values were generated using ω_r=1000, while low machine heterogeneity values used ω_r=10. The ranges were chosen to reflect the fact that in real situations there is more variability across execution times for different tasks on a given machine than the execution time for a single task across different machines.

Different *ETC* matrix consistencies were used to capture more aspects of realistic mapping situations. An *ETC* matrix is said to be *inconsistent* if the *ETC* matrices are kept in the unordered, random state in which they were created. The *ETC* matrix indicates *consistent* characteristics if a machine j executes any task i faster than machine k, then machine j executes all tasks faster than machine k. The consistent matrix can be obtained by sorting every row of the matrix independently. Between two special situations, a *semi-consistent* matrix represents a partial ordering among the machine/task execution times. For the *semi-consistent* matrix used here, the row elements in even columns of row i are extracted, sorted and replaced in order, while the row elements in odd columns remain unordered.

Twelve combinations of ETC matrix characteristics are possible: high or low task heterogeneity, high or low machine heterogeneity, and one type of consistencies (consistent, inconsistent, or semi-consistent). Among the twelve combinations the most heterogeneous environment is modeled with inconsistent, high task and machine heterogeneous *ETC*, and correspondingly the least heterogeneous environment is modeled with consistent, low task and machine heterogeneous *ETC*. Other combinations are between these two extremes. In charts presented in this section, Low and High task/machine heterogeneity are abbreviated to LoLo and HiHi, respectively.

All results reported here are averaged over 50 trials, and done for 200 tasks and 20 machines. The makespan for each experiment is normalized with respect to the benchmark heuristic, which is RC. Unless stated, the learning automata model used in the experiments is L_{RI} with *a=0.01* for algorithm A1 and *a=0.001* for algorithms A2 and A3. For algorithms A2 and A3, the weights λ_η and λ_δ are set to 0.4 and 0.6, respectively, for inconsistent environment, and set to 0.05 and 0.95 for semi-consistent and consistent environments. Matching weightage is set to a smaller value than system utilization weightage in semi-consistent and consistent environments, because in consistent environments all tasks have the same first choice for matching, the fastest machine. There is the same situation in a semi-consistent environment because of its consistent sub-matrix. Therefore, the decisive factor in gaining a better makespan is to maximize system utilization rather than matching proximity. Termination condition is met when, no change in makespan is made for 1500 consecutive iterations, or number of iterations exceeds 500000.

In Figure 1, three proposed algorithms are compared with Min-Min and RC in term of normalized makespan for different heterogeneity and consistency. For inconsistent environment, it can be noted that all three proposed algorithms outperform both RC and Min-Min. For high machine/task heterogeneity, makespan resulted by algorithm A3 is 21 percent less than the makespan resulted from RC. Algorithm A2 performs slightly better than algorithm A1, and algorithm A3 performs better than algorithms A1 and A2. For semi-consistent environment, all three proposed algorithms outperform Min-Min. Algorithms A1 and A3 perform better than RC for high task/machine heterogeneity; however, algorithm A2 fails to outperform RC. Except algorithm A3, the other two algorithms perform worse than RC for low task/machine heterogeneity. For consistent environment, it can be stated that RC and Min-Min performs better than the algorithms proposed in this paper.

Results shown in Figure 2 indicate the fact that the proposed algorithms perform signifycantly better than both RC and Min-Min for inconsistent environments, while they fails to perform better than RC and Min-Min for consistent environment. For semi-consistent environment whose heterogeneity is between consistent and inconsistent, learning automata outperforms Min-Min, but performs very closely to RC. Therefore, proposed algorithms operate better in environments with higher level of heterogeneity.

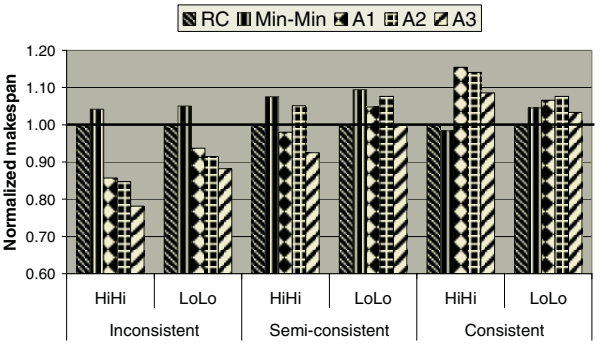


Fig. 1. Comparison of the proposed algorithm with RC and Min-Min for different consistency and heterogeneity

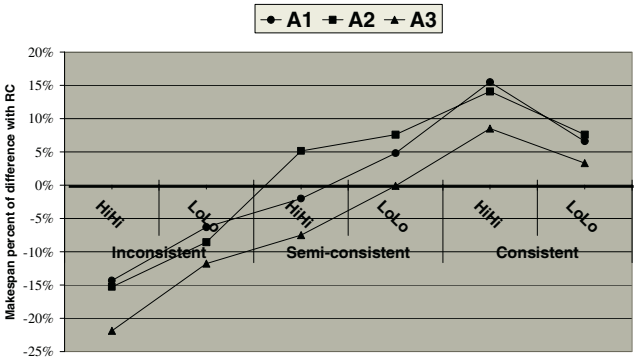


Fig. 2. Difference of makespan with RC for different consistency and heterogeneity

As expected, algorithm A3 performs better than algorithm A2 because it can avoid trapping in local minimums. Observing the results of the experiments, it is evident that algorithm A1 performs very close to and even better than algorithm A2 although it has a completely different reward criterion. It is worth mentioning that in contrast to algorithms A2 and A3 which use detailed information of expected run time of each task on each machine to guide learning automata, algorithm A1 ignores such information and guide learning automata blindly.

The other important issue to consider is the computational cost of finding a mapping using each proposed algorithm. On average, algorithm A2 finds a mapping in about 39000 iterations, while algorithms A1 and A3 needs 12 times more. Setting reward parameter (a) to 0.01 for algorithm A2 but 0.001 for algorithms A1 and A3 may account for faster convergence of A2. However, each algorithm is compared with others by setting learning parameter to a value that yields best result.

7 Conclusion

In this paper, we presented three algorithms based on learning automata for mapping a set of independent tasks over computational grid. The studied computational grid was modeled as a heterogeneous computing environment, and the objective of the proposed algorithm was to assign independent tasks to machines in a way to minimize *makespan*. Through experiments, we showed that for high heterogeneous environments, i.e. inconsistent environments, the proposed algorithms outperform two best existing mapping algorithms.

References

[1] A. L. Rosenberg, Optimal scheduling for cycle-stealing in a network of workstations with a bag-of-tasks workload, *IEEE Trans. Parallel Distributed Systems*, 13(2), 2002, 179-191.

[2] H. Casanova, T.M. Bartol, J. Stiles, and F. Berman, Distributing MCell simulations on the grid, *Int'l J. High Performance Computing Applications*, 15 (3), 2001, 243–257.

[3] S. Smallen, W. Cirne, J. Frey, F. Berman, R. Wolski, M. Su, C. Kesselman, S. Young, and M. Ellisman, Combining workstations and supercomputers to support grid applications: the parallel tomography experience, *IEEE Proc. 9th Heterogeneous Computing Workshop*, 2000, 241–252.

[4] M. Macheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, *J. Parallel Distributed Computing*, 59 (2), 1999, 107–131.

[5] T. D. Braun, H. J. Siegel, and N. Beck, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *J. Parallel and Distributed Computing*, 61, 2001, 810-837.

[6] T. D. Braun, H. J. Siegel, et al., Taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems, *Proc. 17th IEEE Symposium on Reliable Distributed Systems*, 1998, 330-335.

[7] Min-You Wu and Wei Shu, A high-performance mapping algorithm for heterogeneous computing systems, *Proc. 15th Int'l Parallel and Distributed Processing Symposium (IPDPS'01)*, 2001.

[8] K. Narendra and M. A. L. Thathachar, "Learning Automata: An Introduction," Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[9] F. Berman, High-performance schedulers, in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C Kesselman, eds., Morgan Kaufmann, San Francisco, CA, 1999, 279-310.

[10] H. Chen and M. Maheswaran, Distributed dynamic scheduling of composite tasks on grid computing systems, *Proc. Int'l Parallel and Distributed Processing Symposium (IPDPS'02),* 2002.

[11] O. H. Ibarra and C. E. Kim, Heuristic Algorithms for scheduling independent tasks on non-identical processors, *J. ACM,* 24(2), 1977, 280-289.

[12] C. Weng and X. Lu, Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid, *J. Future Generation Computer Systems*, Elsevier, 2003.

Grid Resource Broker Using Application Benchmarking

Enis Afgan, Vijay Velusamy, and Purushotham V. Bangalore

Department of Computer and Information Sciences,
University of Alabama at Birmingham,
1300 University Boulevard,
Birmingham, AL 35294 – USA
{afgane, vvijay, puri}@uab.edu

Abstract. While the Grid is becoming a common word in the context of distributed computing, users are still experiencing long phases of adaptability and increased complexity when using the system. Although users have access to multiple resources, selecting the optimal resource for their application and appropriately launching the job is a tedious process that not only proves difficult for the naïve user, but also leads to ineffective usage of the resources. A general-purpose resource broker that performs application specific resource selection on behalf of the user through a web interface is required. This paper describes the design and prototyping of such a resource broker that not only selects a matching resource based on user specified criteria but also uses the application performance characteristics on the resources enabling the user to execute applications transparently and efficiently thereby providing true virtualization.

1 Introduction

While Grid computing is rapidly evolving and becoming more widely accepted, traditional scientists may still find the use of middleware technologies cumbersome. While the middleware provides effective means to aggregate and virtualize resources, the discovery and categorization of vast resources in this heterogeneous and dynamic environment presents a problem for the end user due to the complexity of the information involved. A general purpose resource broker that facilitates the user's resource selection and job submission automatically is required. While Grid Information Services [11] provides an overview of the available grid resources as well as provides information about the current status of the grid, an average user may be overwhelmed with the information to process, or the user may not have enough experience to select the best available resource. Automation of this selection process would simplify and expedite this process. This Resource Broker was developed as part of the framework available for extension as well as modification that performs application specific resource selection. In order to simplify the process of resource selection and job submission for users, the application's interface is a web-based portlet built as an extension to the Open Grid Computing Environment (OGCE) [15]

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 691 – 701, 2005.

with a simple resource request format. The application is currently being tested to work with mpiBLAST [3]. The primary focus of this research is to design and prototype a resource broker that will not only select a matching resource based on user specified criteria but also use the application performance characteristics on these different resources and enable the user to execute the applications transparently thereby providing true virtualization.

2 Related Work

Resource brokering has been an important area of research for the development of Grid computing. Most research that has been done on resource selection of heterogeneous resources has essentially been from the viewpoint of an application. Condor [13] with its *matchmaker* is another project that is very relevant to the idea of general resource selection. It is based on *ClassAd language*, which allows users, as well as owners of resources, to specify arbitrary restraints. The *matchmaker* is used to match user requests to the available and appropriate resources; in case of multiple matching resources, a ranking system is employed. This ranking system is based on user-specified constraints in order to return the best match.

Nimrod-G [2] is another well-known resource broker. The main functionality provided by the Nimrod-G is automation of creation and management of large parametric experiments [17]. Besides the plain submission of a request for a resource search, users have an option to specify time and cost constraints which are later used in selecting the resource. If the constraints cannot be met, tradeoffs are explained to the user [2,17].

Another well-established resource broker is the Application-Level Scheduler (AppLeS). It is mainly used for scheduling and deploying of parameter sweep applications where tasks have no or little inter-task communication [6]. The main advantage of this resource broker is fault tolerance where any errors are processed and jobs resubmitted on other resources without the need for user intervention[6].

This research focuses on creating a general and easily applicable system that uses application benchmarking. The inspiration behind this application is to bring the resource selection and job submission in the grid to a practical level with user friendly orientation. Incorporation of the resource broker into the OGCE achieves the goal since the installation of the entire system is simple and part of a single package. Once the package is installed within a virtual organization, it can be accessed without any additional user-end configuration. Other approaches have not been completely integrated into a single yet complete system designed for easy installation and access.

3 The Resource Information Problem

The resource selection process is based on information available about a resource. Unlike the local resource schedulers (*e.g.,* PBS [18], LSF [21], LoadLeveler [10], SGE [8]) that implement fine-tuned scheduling policies based on resource requirements running on the nodes as well as those waiting in the queue, resource selection and application-level scheduling in the Grid has a limited amount of

information available. All the information is available from Grid Information Services (GIS or MDS) [11] provided by Globus [4]. This provides a way to discover current system information, such as the static configuration of a computer as well as the current load and status.

Some of the criteria used for resource selection at the application level include:

- application-specific requirements
- static resource capabilities
- dynamic state of the resource

This information needs to be collected individually and subsequently combined and processed into one meaningful result. Our resource broker conforms to this model, as is discussed throughout the rest of the paper.

One more aspect worth noting is that any form of the application-level resource selection should be done based on the resource consumption in terms of that application. As an example, a heavily loaded resource from the view point of one application might be just the opposite for another application. This creates a wider pool of available resources as well as gives the resource selection process more options during the selection process. Consequently, in the context of limited information available in the Grid environment described earlier, the difficulty of writing a general purpose, yet efficient, scheduling algorithm increases.

4 Architecture

This section describes the integration of the resource broker with the OGCE, why we chose such an approach, as well as discusses resource broker architecture in detail.

4.1 OGCE and Resource Broker

OGCE [15] is a portal developed to provide easy access to Grid technologies (*i.e.,* through the web-based interface). It provides sharable and reusable components for web-based access to scientific and business-oriented applications. Sharable components make it easy to quickly create Grid Portals from provided libraries that support baseline Grid technologies, such as file transfer, job launching and monitoring, and access to information services. This frees the developers to concentrate on the specialized needs of a particular scientific community [15]. Although GridSphere [14] could also be used for the portal, since OGCE is an NMI supported project , we have chosen to use OGCE for this architecture.

Since OGCE was, mainly, designed and developed to hide the basic Grid infrastructure, a resource broker was not implemented because it is not part of the capability offered by Grid base services (*e.g.,* MDS). Addition of the Resource Broker (RB) to OGCE adds a new layer of abstraction between the user and the Grid while it uses many base grid services, thus working toward the idea of seamless job submission. OGCE was built using J2EE technology to create multiple portlets representing different grid services. The resource broker adds a new service to the system using the same approach.

4.2 Architecture of the Resource Broker

The architecture of our resource broker itself is shown in Figure 1. The resource broker consists of four main components, each having the basic functionality of providing standard interfaces to the rest of the application. Each of these components is discussed next.

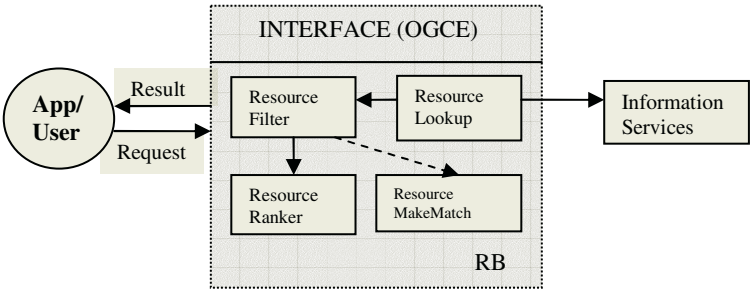


Fig. 1. Resource Broker architecture showing main components of the design

4.2.1 Resource Lookup

Is a module that acts as a front-end to the information services. Information available from Ganglia [7] is used for this purpose. Ganglia is a scalable distributed monitoring system for high-performance computing systems [7], and is, in turn, very similar to MDS [11] provided by Globus [5]. The major difference between Ganglia and MDS is the security aspect - MDS is configurable to use GSI [4], while Ganglia is an unshielded system. Any information retrieved from Ganglia should also be available from MDS and transformation from using one system to the other is easily supported by our resource broker. The Resource Lookup component is responsible for connecting to the information services and extracting the information for use by the RB. It generates a representation of the data about the available resources that is internal to the RB, thus, providing standard interface and format for the rest of the RB to use during the extraction of the data. Use of this interface module as a data reader makes use of different or, even, multiple information service providers. This simply requires a quick adaptation. The Resource Lookup does not do any sort of data organization or processing, other than storing it in the internal structure as it was received by the information source.

4.2.2 Resource Filter

Presents an effort to create, again, a generic functionality that, among others, implements a resource broker specific *compareTo* method which is specifically designed to compare the user request to the available resources. It provides a standard interface where the implementation of the selection process can be easily adapted for various formats of user request. In the current implementation of the resource broker, Resource Filter performs the basic filtering function of rejecting any non-matching string variables such as the operating system type, as well as the type of requested architecture. The idea of filtration was further extended to numerical-valued

components. Rather than just performing a basic true/false match for each component, a system of weights is employed that gives more insight into how a resource compares to the requested values. The weights represent how a requested value of a component compares to the resource capability in terms of percentage. These values are later used in the ranking of the different resources. Another function performed by the Resource Filter is subdivision of compared results in fully matching resources and non-matching resources. The idea is to distinct but not throw away the non-matching resources since there might be some that are very near the request, or applicable to an application, which is determined from the weighted system in the MakeMatch component.

4.2.3 Resource Ranker

Is the most important single piece of this application. It is designed to rank multiple matching resources and return the most appropriate resource for the user-submitted application. Typically, this is challenging to design and implement due to the described lack of resource information available, as well as the complexity of the brokering algorithm employed. This component has two general parts: information collection and information processing. In the first of the three stages of the information collection component, a rough resource selection, primarily based on hardware requirements supplied by the user, is performed from a pool of all the available resources. This is followed by collecting application-specific performance information from individual resources. In the final step, this section processes each resource, one component at a time, applying a value function that takes into consideration the user's rank of the individual component. Following this initial information collection, a cumulative resource rank value is calculated in the information processing section. In an attempt to generate rank values that are not based only on the static values of a resource, a dynamic load function is applied which combines its prediction values with the results from the second step in the information collection part. The essential idea behind the second step in the information collection section is to submit a job to each of the matching resources from step one using a sample problem set and then monitor some performance meters. Currently, this performance measuring process is done by brute force for each individual job submission, but there is work being done on individual application profiling which would be used to help in this step resulting in more accurate and prompt scheduling policies. The benefits of processing resources at the level of individual components allows for application oriented resource selection. Initially, this orientation is user-dependant through the ranking values of the components, but as described in the future work section and in combination of the mentioned application profiling, a system will be provided where the selection of the best resource for the given application will be done by the resource broker itself.

4.2.4 Resource MakeMatch

Is an implementation of a technique that comes in very useful in the case where not a single full match is found, or even when there is just a small number of fully matching resources. There are three preferences the user can customize regarding this procedure. First, the user is given the option of selecting the minimum number of resources that must be fully matched before this system is invoked. The second option

is for the user to specify a range of values for which a request is valid, and lastly, the user has the option to rank individual components as they are pertinent to the submitted application. In the case where no fully matching resources are found, this system is invoked by default in order to suggest some possible matches. The way this system works is that each of the non-matching resource's individual component's weighted values, as computed by the Resource Filter, are tested against the user-specified rank and/or range of possible values. Once all of the components are processed a custom rank for each resource is generated. This is used by the ResourceRanker in picking the most appropriate resource. This functionality was added for two reasons; one is to make user submissions simpler by increasing the chances of a match, and the second was inferred from [20]. In this paper, the authors point out that resources with weaker capabilities generally have a smaller variance and, thus, provide a better base for predicting the future load, resulting in more accurate scheduling procedures. This implies that a resource that might not appear to be adequate for the job may, in the end, produce a useful result, and the resource broker is used to suggest some of those resources.

Division of the work done by the resource broker in the manner just explained creates a small framework that can be used as a base for plugging in other components or replacing the current ones while maintaining the main functionality. An obvious possibility is the expansion of this resource broker into a scheduler once the advance scheduling becomes more developed [9]. This design is intended to provide a set of API to facilitate this extension.

4.3 User Request

A request includes the following two elements, apart from the instructions on how to run the application:

- *Resource description*: user requirements of the required resource, such as CPU speed, available memory, type of operating system
- *Individual component ranks*: for each component in the Resource description, a weight signifying the importance for that component's full match

Figure 2 shows a sample request to describe the options available for the user:

```
/1/  cpuCount = 4                  /4/  cpuCountRank = 10
/2/  cpuSpeed = 2394               /5/  cpuSpeedRank = 6
/3/  cpuType  = Pentium IV         /6/  cpuTypeRank  = 8
                …                                  …
```

Fig. 2. Partial showing of a sample user request

As the work in Global Grid Forum continues to approach the standard of the job description language [12], the simple request structure will allow us to use it in specifying job requirements more completely.

4.4 Brokering Algorithm

The brokering algorithm, or information processing section, is the second part of the *ResourceRanker* component. It is subdivided into two parts: resource rank value calculation and application of the load function. Figure 3 describes the algorithm.

In order to calculate the initial resource rank value, each matching resource is processed on a per component basis taking into consideration user's rank values for all the components. The resulting value relates each of the resources individually as well as to the initial request. A separate rank value, for each resource again, is calculated based on the performance measure of individual resource. The load function, supplied by the two initial resource rank values, considers current load and does prediction of future load based on load variance over the past 15 minutes, as provided by the information services. Both of the load functions are using fuzzy logic [19] with Fuzzy Engine for Java [16] as the fuzzy logic engine.

For each resource,
- a. For each component,
  - i. Calculate rank value taking into consideration user rank specification
- b. For each resource,
  - i. Calculate rank value based on application-performance measure
- c. Apply load function to adjust resource rank value
  - i. Consider load variance and use it for future load prediction
  - ii. Consider current load value

Fig. 3. Brokering Algorithm for the generic resource selection

The load function for considering the change in load variance employs six different membership functions, ranging from high positive change in the load to the high negative change. The fuzzy engine uses the trapezoidal membership function to determine the degree to which the load change belongs in a group. The parameters for each of the membership functions are statically assigned for now, but hope to soon turn this system into a self-learning one where these parameters can be automatically adjusted as more and more jobs are submitted using this resource broker.

The current load value function uses a simpler membership function set, as well as fewer rules. Since it is using only one input variable (*e.g.,* load value), it has three membership functions and three rules that control the outcome. The membership functions range from low to high again with statically typed parameters.

Fuzzy logic is used in this part of the application, primarily since it allows for an easy and, automatic, way to dynamically assign the same load value to different membership groups. Depending on different user applications, as well as when applied in relation to systems connected to the grid that do not belong in the computationally intensive category, different parameters for resource selection should be used. Using fuzzy logic is an elegant way to allow for these changes. As this system evolves, we foresee a user option to select the type of application and/or resource they have or require in order to use this system in the best possible way.

5 Application Deployment Case Study

Instead of measuring the time saved or time spent by the resource broker, we present a use case outlining the major steps. The application we use in our simulation and test case is mpiBLAST [3]. It is a tool used for sequence analysis and interpretation in genomic sequencing. This being a popular application among bioinformatics researchers, it is an important and excellent gateway for the resource broker to bring an existing cluster application to a Grid application. For our test case, we are assuming the application is installed on all of the remote resources and all the necessary files needed to run the application are available in a user accessible directory.

Table 1 compares the steps for job submission via the resource broker versus using command line tools. Italicized steps are automatically done by the resource broker.

Table 1. Resource Broker vs. Manual Grid Job Submission

Resource Broker	Manual Job Submission
0. The user has to submit their credentials to MyProxy server. This is server administration dependant, but generally, the credential should be renewed once a month. 1. The user submits the request supplying desired resource information along with the parameterized command on how to run application (*e.g*. mpiBLAST) 2. *The Resource Broker queries GIS.* 3. *The Resource Broker selects the best available resource.* 4. *The Resource Broker acquires user's credentials.* 5. *The Resource Broker submits the job to a resource using user's credentials.* 6. *The Resource Broker transfers any output files.* 7. The user can monitor job progress from the webpage. 8. The user is notified when the output files have been transferred back to the user's local machine.	1. The user must have a valid user certificate on the local machine and request a user proxy from GSI. 2. The user must query information services, (GIS/MDS) which send back an XML formatted document listing all the information about all the known resources. 3. The user must process the returned information and select the most appropriate resource for their application. 4. The user must submit the job to the selected resource. This involves using authentication, command line job submission (*i.e.* create RSL command) and making sure the application is submitted correctly. 5. The user can monitor the application progress using command line tools. 6. Upon the end of the run for the application, the user should retrieve any output files back to the local machine using GridFTP [1]

6 Future Work

The main focus of future work is to integrate this infrastructure with the university grid and facilitating its users. User and application profiles will be stored in a database for easy retrieval. The most interesting part of the application information retrieval involves application profiling which would gather run time information about specific

applications on given resources. This information would enable the brokering algorithm to be enhanced and to make use of the precise past information when performing the selection. This specifically refers to the parameters used in the fuzzy logic components of the load function. These parameters currently have single, statically typed values which will be modified in stepwise fashion in the following two ways. The first option is to have different initialization procedures, depending on the application submitted, so that the parameters reflect application-oriented scheduling. The selection of different parameter options will be left to the user, based on their knowledge of the application. Another option regarding the modification of the parameter values refers to the adjustment of the values within an application group. Through the use of application profiling an automatic learning method will be implemented which, after the initial learning stage, would automatically monitor jobs for each resource selection, and adjust the parameter values.

In the future, a set of API will be made available for extension. We hope to extend this application into a full grid access system that is easy to use and yet powerful in any given environment, *e.g.,* a co-scheduler which could be used to schedule jobs in the long run among the selected resources.

As work on the Job Description Submission Language (JSDL) evolves to a usable standard [12] the user request format would also evolve from one specifying a resource in terms of its components to one describing the application itself.

7 Summary

This paper introduces a resource broker that bridges the gap between a user finding of a resource, and job submission. Athough manual operations could achieve the same goal, it would be time-consuming and complex. This resource broker was developed in order to make the transition to using Grid technologies simple and efficient.

This is a general-purpose resource broker with a simple and understandable interface providing appropriate resource selection capabilities for different types of applications. It is an attempt at developing a small framework where custom or tested components can be added as well as replace current ones. The current scheduling algorithm works based on information retrieved from MDS as well as performance measure of submitted application. It uses fuzzy logic during the resource profiling which would easily adapt to multiple user requirements based on different types of resources and jobs. We have tested our application with the mpiBLAST to validate our architecture design as well as the brokering algorithm. Preliminary results are promising, especially with respect to usability.

To the best of the authors' knowledge this study is the first investigation of incorporating application profiling into the resource scheduler, in order to simplify the selection, usage and enable efficient utilization of resources. Future plans include testing and adopting more applications, incorporating a metascheduler, and including application profiling with feedback.

Acknowledgments

The authors would like to acknowledge their colleagues from High Performance Computing Laboratory, specifically Zhijie Guan for providing support as well as

insight into some of the problems faced during this work. This work was in part supported by The Department of Computer and Information Sciences at the University of Alabama at Birmingham.

References

1. Allcock, W., Bester, J.Bresnahan, J., Chervenak, A., Liming, L., and Tuecke, S., *Draft GridFTP Protocol*, 2001, [Last accessed, Available from http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf.
2. Buyya, R., D. Abramson, and J. Giddy, "Nimrod-G: An Architecture for a Resource Management and Scheduling in a Global Computational Grid", In *Proceedings of 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000)*, at Beijing, China, May 14-17, 2000.
3. Darling, Aaron E., Lucas Carey, and Wu-chun Feng, "The design, implementation and evaluation of mpiBLAST", In *Proceedings of ClusterWorld Conference & Expo in conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution 2003*, at San Jose, CA, 2003.
4. Foster, I., C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids", In *Proceedings of ACM Conference on Computer and Communications Security*, ACM Press, at San Francisco, CA, 1998, pp. 83-92.
5. Foster, Ian and Carl Kesselman, The Globus toolkit, In *The Grid: Blueprint for a New Computing Infrastructure*, Chapter 11, Edited by Foster, Ian and Carl Kesselman, pp. 259--78, San Francisco, California, 1999.
6. Fran, B., W. Rich, F. Silvia, S. Jennifer, and S.Gary, "Application-Level Scheduling on Distributed Heterogeneous Networks", In *Proceedings of Supercomputing '96*, ACM Press, at Pittsburgh, PA, 1996, p. 28.
7. *Ganglia*, 6/1/2004, 2004, [Last accessed 6/15, 2004], Available from http://ganglia.sourceforge.net/.
8. Gentzsch, Wolfgang, "Sun Grid Engine: Towards Creating a Compute Power Grid", In *Proceedings of Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID '01)*, IEEE Computer Society, 2001, pp. 35-6.
9. *Grid Scheduling Architecture Research Group*, 2004, [Last accessed 6/15, 2004], Available from http://forge.gridforum.org/projects/gsa-rg.
10. "IBM LoadLeveler: User's Guide", International Business Machines (IBM), September, 1993.
11. *Information Services/MDS*, 6/14, 2004, [Last accessed 6/15, 2004], Available from http://www.globus.org/mds.
12. *Job Submission Description Language Working Group (JSDL-WG)*, 3/29, 2004, [Last accessed 6/15, 2004], Available from http://www.epcc.ed.ac.uk/%7Eali/WORK/GGF/JSDL-WG/.
13. Litzkow, M., M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations", In *Proceedings of 8th International Conference of Distributed Computing Systems*, June 1988, pp. 104-11.
14. Novotny, J., M. Russell, and O. Wehrens, "GridSphere: A Portal Framework for Building Collaborations", In *Proceedings of 1st International Middleware Conference*, at Rio de Janeiro, Brazil, June 16-20, 2003.

15. *OGCE - Open Grid Computing Environments Collaboratory*, 1/22, 2004, [Last accessed 6/15, 2004], Available from http://www.ogce.org/index.php.
16. Sazonov, E. S., *Open source fuzzy inference engine for Java*, [Last accessed 6/15, 2004], Available from http://www.clarkson.edu/~esazonov/FuzzyEngine.htm.
17. Steen, Martin van, *Nimrod-G Resource Broker for Service-Oriented Grid Computing*, 2004, [Last accessed 6/15, 2004], Available from http://dsonline.computer.org/0107/departments/res0107_print.htm.
18. Systems, Veridian, OpenPBS v2.3: The Portable Batch System Software, 2004.
19. The MathWorks, Inc, *What Is Fuzzy Logic?* , 2004, [Last accessed 6/15, 2004], Available from http://www.mathworks.nl/access/helpdesk/help/toolbox/fuzzy/index.html.
20. Yang, L., J. M. Schopf, and I. Foster, "Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments", In *Proceedings of Super Computing 2003*, ACM Press, at Phoenix, AZ, 2003.
21. Zhou, Songnian, "LSF: Load Sharing in Large-scale Heterogeneous Distributed Systems", In *Proceedings of Workshop on Cluster Computing*, 1992.

The Grid Block Device: Performance in LAN and WAN Environments

Bardur Arantsson* and Brian Vinter

IMADA, University of Southern Denmark,
Campusvej 55,5230 Odense M, Denmark
Tel.: +45 6550 2387 Fax: +45 6593 2691
{bardur, vinter}@imada.sdu.dk

Abstract. We present initial results from the implementation of the Grid Block Device, a distributed, replicated block device for the Grid based on the the replication algorithm of Y. Amir[1]. It supports application-specific replication strategies and consistency models. Although the presented WAN results are dissapointing because of issues with the underlying group communication library, we believe the LAN results show that the Grid Block Device could become a viable solution for truly distributed storage.

1 Introduction

Achieving good scalability using traditional cluster configurations and current commodity hardware is becoming increasingly difficult as the storage demands of scientific applications continue to soar. New "superclusters" made up of large clusters connected via dedicated high-speed Wide Area Network (WAN) links are also becoming increasingly common, and while these decicated links have high throughput and have relatively low latencies, the basic physical limit of the speed of light sets a fixed lower bound for the best possible latency. Thus, truly distributed storage solutions must be latency-tolerant.

The Grid Block Device (GBD) has been developed to provide simple and efficient fault tolerant storage with many desirable features, among them replication and disconnected operation. The reason we have chosen a block device as the abstraction instead of a file system is simplicity and efficiency: File systems imply certain semantics which many Grid applications do not actually require.

This paper shows that the prototype GBD implementation can achieve quite reasonable performance in LAN environments.

2 Previous Work

While many distributed file systems are currently being developed (see e.g. OceanStore[2], Mammoth[3] and Lustre[4]), they all focus on the data distri-

* The work of this author is sponsored by the Faroese Research Council.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 702–710, 2005.

bution problem at the granularity of files/objects, whereas we focus on it at the granularity of fixed-size blocks. There are also systems, e.g. COMA [5], which implement distributed paging which is similar to the GBD model, but they are often very specific to a particular overall system architecture. Moreover both of these types of systems often have built-in semantics which may be needlessly strict or overly lax for a particular application.

2.1 Spread

The low-level communication layer utilized by the replication algorithm is the Spread [6] group communication system. Spread provides efficient group communication with reliability and ordering guarantees. To reduce end-to-end latency and provide reliable efficient multicast services with ordering, it uses UDP/IP as the underlying communication protocol and implements the end-to-end reliability and ordering on top of that.

2.2 Replication Algorithm

The basic replication algorithm used in the GBD is a slightly modified version of Y. Amir's algorithm in [1]. It is based on the Extended Virtual Synchrony [7] concept and provides fault tolerant replication whilst considerably reducing the need for – and for all common cases avoiding – the most expensive part of most other replication algorithms, namely end-to-end acknowledgements. Avoidance of end-to-end acknowledgements on a per-update basis can yield much higher performance than algorithms such as 2PC and COReL as demonstrated in [8]. Network partitions are also handled correctly by the algorithm.

The replication algorithm is designed mainly for database-like scenarios where the goal is to replicate the complete contents of a database to all participating servers. Instead of taking the traditional approach of replicating the data itself to participating servers, it instead replicates all *actions* taken upon the data, and ensures that all these actions are applied across all replicas in a global total order. While ensuring that all participating servers have functionally identical replicas, this also affords a lot of flexibility, and, in particular, permits using different consistency semantics for different actions.

Originally, the replication algorithm in [1] was restricted to a preconfigured set of replication servers, but it was extended in [8] to allow for dynamic addition/removal of replicas.

3 Overview

The overall architecture of the Grid Block Device is very simple (see figure 1); it simply sits as a layer between the replication/semantics engine and the client application. It presents the application with a block device abstraction which simply happens to be distributed and replicated semi-transparently. Note that the "application" is not necessarily a user application, but could just as easily be a distributed file system implementation running on top of the GBD.

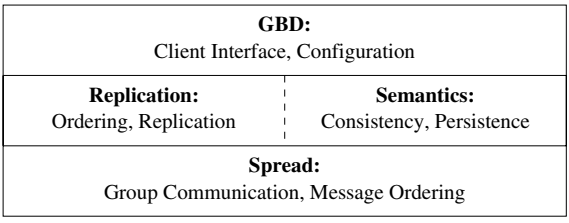


Fig. 1. GBD overall architecture

In the following sections, we will motivate and discuss the implementation of the features of the GBD in relevant detail.

3.1 Distributed Servers

Having geographically distributed servers is critical to high availability and low latency for widely distributed clients accessing the system. Using widely distributed servers has traditionally been problematic because the inter-server communication has usually been done using protocols and algorithms (e.g. 2PC) with a lot of latency-sensitive overhead, or, which simply could not operate correctly in partitioned networks. Clearly, network partitioning is to be expected in any reasonably-sized WAN, simply because of the distances involved.

Since the algorithm of [1] can operate in partitioned networks and avoids per-update end-to-end acknowledgements, it should be relatively cheap to employ widely distributed servers in this context. However, there may be a tendency for more conflicting updates to occur in such systems, so the result may be an increase in the number of failed updates.

3.2 Replication and Migration

The ability to replicate and migrate data to where it is needed is among the most important features of the Grid Block Device. Much research has been conducted in this area; see [9] for an overview of early work. There are also some randomized algorithms for the replica placement problem that are quite simple to implement which also do well in practice; see [10], for example.

One problem with choosing among these algorithms is that the underlying assumptions can vary wildly, and may or may not apply equally well to all applications. Clearly, the optimal choice of algorithm may also vary from application to application.

Instead of trying to dynamically pick an optimal replication/migration strategy based on application access patterns we have simply chosen to let the application decide for itself when it wants to introduce/remove replicas or migrate blocks of data. Therefore, the GBD contains explicit primitives to initiate replica migration/removal.

3.3 Disconnected Operations

In general, we distinguish between two forms of disconnected operation, namely *involuntary* disconnection and *voluntary* disconnection, but the GBD does not currently distinguish between the two.

Keeping all the replicas consistent is, of course, the primary concern when disconnected operations are permitted. When the network configuration changes, the replication engine automatically selects a new primary component which is allowed to continue unhindered. However, for any servers that are not in the primary component, special care must be taken to keep data consistent. Depending on the consistency requirements specified by an application, an operation is either performed immediately or delayed until the desired consistency can be achieved.

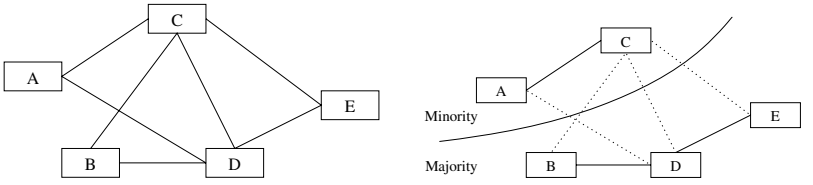
Mitigating the impact of such disconnection events on the performance of the rest of the network is, of course, of great importance. Obviously, there are instances where a performance degradation simply cannot be avoided, the most extreme case being when network becomes partitioned and all servers with a particular datum are isolated from the rest of the network (see figure 2).

The GBD provides primitives to introduce both regular replicas and what we call *transient replicas*. Transient replicas are similar to regular read-only replicas, but for any particular set of connected nodes, the GBD only guarantees the consistency of a transient replica within that set. Among other things, this means that it is possible to create transient replicas for data without being able to reach the majority of the network – such replicas will naturally only be weakly consistent with respect to the whole network. To see why this is useful, imagine the following scenario: If a set of servers with, say, 1 replica of a particular datum becomes disconnected from the majority, and the majority subsequently fails, the total replica count is reduced to 1. However, the disconnected component cannot know this until the network becomes whole again, so the data remains vulnerable to failure until that happens. By introducing transient replicas in response to a component becoming disconnected from the majority of the network, the replica count can be temporarily increased until the system can ensure that enough replicas exist.

3.4 Global Snapshots

Supporting global snapshots in distributed system is usually not trivial by any means, because snapshots have very precise and strict semantics: A snapshot represents a *consistent* and *unchanging* image of the data, in our case the contents of the block device, at the time the snapshot was taken. Global snapshots can be used for a variety of things, most commonly to allow backups in systems which cannot be taken off-line.

There are two basic techniques for supporting snapshots: 1) Create a read-only duplicate of the data, or 2) avoid overwriting data that is part of a snapshot and write new copies of modified data instead. Since the former has steep storage requirements and requires delaying all updates until a complete copy has been made, we have chosen to use the latter approach. It requires some extra



(a) Fully connected network. (b) Partitioned network.

Fig. 2. Effects of network partitions: If node A and node C have some datum, x, which is required by node B, D or E, then a network partition may prevent all access to x until the network becomes whole again. However, the majority may continue issuing and performing updates on shared data without causing consistency problems

bookkeeping, but its advantages far outweigh its disadvantages. Firstly, a snapshot only requires storage on the order of the number of blocks modified after the snapshot was taken, and secondly it performs the minimal amount of copying needed. Implementing it is simply a matter of keeping track of the active snapshots and storing extra copies of blocks which are subsequently overwritten.

4 Performance Testing

In order to test the basic throughput of the GBD system compared to just using the raw Spread group communication protocol, we have performed a couple of simple benchmarks. The idea is simply to connect to the GBD or Spread and try to write/send a fixed amount of data. Measuring the time taken, we get the throughput.

To avoid the potential bottleneck from disk I/O, all disk-related activity was turned off during these tests.

The performance test results presented here were all obtained on a single pair of identical Pentium 4, 2.4GHz machines with a switched 1GBps Ethernet connection.

It should be noted that the GBD is implemented in OCaml, an efficient high-level functional language.

4.1 LAN Performance

The benchmark yielded the results shown in figure 3. The Spread/C plot shows the results of running the purely C-based benchmark, and all the other Spread results were obtained from OCaml-based benchmarks. The text in parentheses indicates the method used for actual communication: "blocking" indicates that blocking reads/writes were being used, "engines" indicates that the object-oriented *Equeue* [11] non-blocking I/O framework was being used, and finally, "select" indicates that a raw `select()` loop was being used. It should be noted

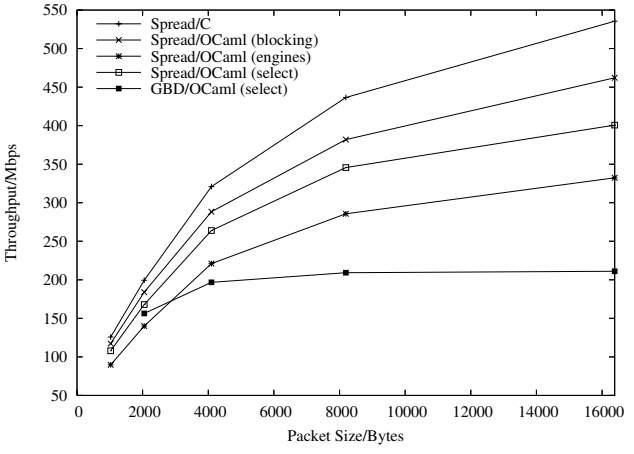


Fig. 3. Throughput as a function of packet size on our two test machines. The Spread graph shows the throughput using only Spread for communication, and the GBD graph shows the throughput when using the GBD on top of Spread

that the GBD requires some form of non-blocking I/O because it needs to be able to communicate with clients while also communicating with the Spread daemon.

There is one obvious issue which contributes to the GBD overhead seen in the plot: To guarantee proper update ordering in the GBD, all client requests must be "repackaged" and sent to the whole group of servers (see figure 4). All communication in this particular scenario is handled using TCP, which, given adequate buffering, performs quite well for our purposes. It may seem odd that the Spread daemon and the GBD server are running on separate hosts where one might expect them to be running them on the same host (thus avoiding superfluous network traffic). The explanation is simply that this yields better performance than having both the Spread daemon and the GBD server on the same host (due to the rather steep CPU requirements of Spread).

Contrasting this with figure 5 which shows the pure Spread benchmark we see that it effectively means that this benchmark needs to push twice the amount of data through the Spread daemon.

It is therefore not surprising that the GBD performance reaches its maximum of around 200Mbit at roughly 50% of the Spread maximal performance which is around 400Mbit (when using `select`).

4.2 WAN Performance

In figure 6 we see the results for our WAN test. Because of practical issues these tests could only be run on a "simulated" WAN connection using the same equipment as in the LAN tests, and using the Linux kernel's `sch_netem` [12] packet scheduler to delay packets between the two machines by the specified amount (with appropriate jitter and packet loss parameters).

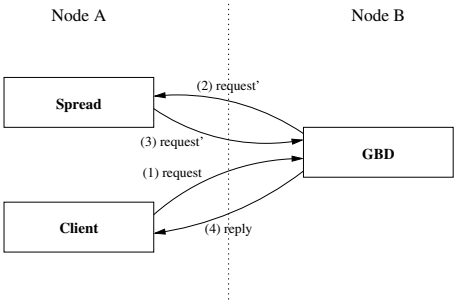


Fig. 4. The path of a request travelling through the GDB system. Repackaging occurs between step 1 and 2

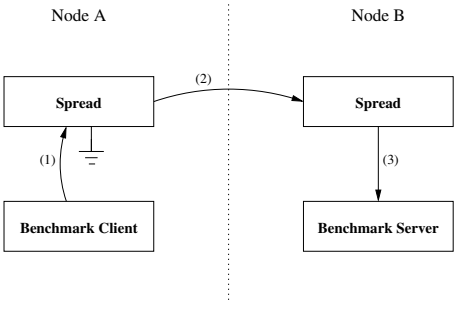


Fig. 5. The path of a request travelling through the Spread benchmark setup. The Spread daemons communicate using UDP, everything else uses TCP. The "ground" indicates that the message is not received by the sender itself

As we can see the results are very dissapointing, and suggest that the Spread protocol, which is token-based, sends the token back and forth far too often without "accumulating" enough data to enable any kind of reasonable throughput in a WAN.

However, we believe that the Spread developers are currently working on improving Spread WAN performance, so there is no reason to believe that these results could not be improved drastically.

4.3 Future Work

Having found a baseline for the achievable performance we are currently benchmarking more interesting scenarios, e.g. the cost of having multiple replicas, the cost of reestablishing replicas after failure, etc.

Clearly, there is also considerable room for improvement when running in WANs, but we believe that the performance problem lies with the Spread implementation and not the GDB. There are two obvious solutions to this: 1) Either we implement our own group communication system with sufficiently strong

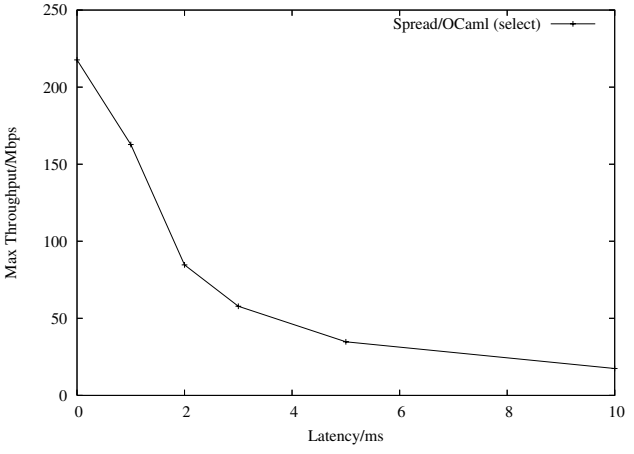


Fig. 6. Throughput as a function of simulated latency between our two test machines. There are no results for GBD as it would quite obviously be pointless to include them. We have only shown the maximum throughput for each latency value since there was almost no variation for different packet sizes (except at the very low end of the latency scale)

message delivery semantics, or 2) we simply wait for the Spread developers to improve WAN performance.

5 Conclusion

The benchmarks have shown that we can acheive quite reasonable performance in a 1Gbit LAN, although we have not yet shown this to be the case for non-trivial setups.

Unfortunately, the WAN tests are very dissapointing because of the lack of Spread optimization for high throughput in WANs, but work is underway to optimize Spread for this scenario. Esssentially the GBD does not rely on any end-to-end communication during normal operation and *should* be quite capable of high throughtput while maintaining consistency even in situations with high latency, though we cannot show this currently.

References

1. Amir, Y.: Replication Using Group Communication Over a Partitioned Network. PhD thesis (1995)
2. Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: OceanStore: An architecture for global-scale persistent storage. In: Proceedings of ACM ASPLOS, ACM (2000)

3. Brodsky, D., Brodsky, A., Pomkoski, J., Gong, S., Feeley, M., Hutchinson, N.: (Using file-grain connectivity to implement a peer-to-peer file system)
4. (http://www.lustre.org)
5. Saulsbury, A., Wilkinson, T., Carter, J., Landin, A., Haridi, S.: An argument for simple COMA. Technical Report R94:15, Swedish Institute of Computer Science (1994)
6. Amir, Y., Danilov, C., Stanton, J.: A low latency, loss tolerant architecture and protocol for wide area group communication. In: FTCS 2000. (2000)
7. Moser, L.E., Amir, Y., Melliar-Smith, P.M., Agarwal, D.A.: Extended virtual synchrony. In: The 14th IEEE International Conference on Distributed Computing Systems (ICDCS). (1994) 56–65
8. Amir, Y., Tutu, C.: From total order to database replication. Technical report, Johns Hopkins University (2002)
9. Dowdy, D., Foster, D.: Comparative models of the file assignment problem (1982)
10. Bartal, Y.: 5. In: Lecture Notes in Computer Science 1442. Springer (1998)
11. Stolpmann, G.: Equeue. (http://www.ocaml-programming.de/programming/equeue.html)
12. Hemminger, S.: Network emulator. (http://developer.osdl.org/shemminger/netem/)

WS-Based Discovery Service for Grid Computing Elements

Kazimierz Balos and Krzysztof Zielinski

Department of Computer Science,
AGH University of Science & Technology,
Al. Mickiewicza 30, 30-059 Krakow, Poland
{kbalos, kz}@agh.edu.pl

Abstract. Monitoring system for grid requires an easy solution for clusters location and registration. The paper describes discovery service for Java Management Extensions and Web Services based infrastructure monitoring system which solves this problem. System was designed and put into practice in existing installation of development grid in EU IST CrossGrid project. Described discovery service for Computing Elements didn't affect the installation and operation of existing monitoring system. Durability, immunity to failures of clusters, and no additional firewall configuration requirements are the main features of the service. In Section 1 there is presented the concept of WS-based Global Discovery Service (WS-GDS). Section 2 presents features causing that this solution fits best to existing monitoring system based on WS and JMX. Section 3 contains WS-GDS sequence diagrams. Section 4 covers the implementation issues of GDS module. Section 5 sums up the advantages and disadvantages of accepted solution and compares it with the other systems and frameworks that could be used. The paper is ended with conclusions.

Keywords: Grid, discovery service, Computing Element, monitoring, JMX, MBean.

1 Introduction

Monitoring systems operating in particular clusters in grid networks need to be registered in central registry in order to be able to provide the list of all available systems. In environments where cluster and grid topology changes very often, it is required to keep actual list of monitoring services and their access points. Assuming that every cluster has one Web Service acting as a SOAP gateway for accessing monitoring parameters [6,7], there should be provided service that can deliver actual and valid list of all operating SOAP gateways that can be accessed by consumer at any time.

There have to be fulfilled several conditions before design of Global Discovery Service can be adopted in particular monitoring system, especially in system applied in CrossGrid project, based on JMX (Java Management Extensions) [1,2] and Web Services [5]. These conditions include:

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 711 – 720, 2005.

1. fail over, i.e. immunity to failures of SOAP gateway registry,
2. accuracy - invalid entries should expire after certain period of time,
3. possibility of applying the solution in current monitoring system without changes in firewalls and restarting the whole monitoring system.

Global discovery service of all services running in different clusters could be based on a collection of chosen gateways. These gateways can act as a registry of SOAP gateways beside their normal function of registering local Worker Nodes. For fail over purposes the number of such machines can be increased to two or more. In extreme case it can be equal to the number of all running gateways.

Possibility of practical application of presented solution is not less important than other functional features. Described solution takes advantages of current configuration of monitoring system and uses the available entry points to clusters in order to register appearing services. Though monitoring services and their interfaces based on Web Services can be considered rather read only, it's possible to use the same mechanism to treat SOAP gateways as chosen GDS nodes to perform the registration and discovery. The way of calling a method for reading some monitoring parameters can be used for setting some parameters, and also for new services registration. It won't affect the configuration of firewalls, and will be homogenous with other services exposed as Web Services.

First part of the article presents layered architecture of JIMS – the JMX-based Infrastructure Monitoring System, briefly describes all layers, and finally defines the problem of discovery service. Next section shows WS-GDS sequence diagrams to show the CE (Computing Element) search process. Implementation of this service using Web Services and JMX is described in Section 4. There are shown elements of GDS module (implemented as JMX MBean [1]): its all attributes and methods. Section 5 covers the WS-GDS module evaluation and defines metrics that can be used for its performance estimation. Short discussion of other possible solutions, their advantages, disadvantages, and security concerns, is included in the conclusions.

2 WS-Based Global Discovery Service

Presented system of Computing Elements registration was implemented as a module for JIMS - monitoring system used in EU IST-2001-32243 CrossGrid project. **Fig. 1** shows layered architecture of the system. The lowest layer provides resources instrumentation and is responsible for making available information concerning grid resources such as CPUs, memory and network using:

- virtual file system (/proc) and Java Native Interface (JNI) for Linux kernels 2.4.x and 2.6.x monitoring modules,
- SNMP version 2c agents running locally in monitored systems,
- other modules, for example dedicated network module (NetworkMetrics) for ICMP and UDP measurements of latency and available bandwidth between monitored nodes [10] or Sun Grid Engine [4] monitoring module.

Second layer, the interoperability layer, provides the common point of communication with all worker nodes (WNs) in cluster through dedicated node – CE.

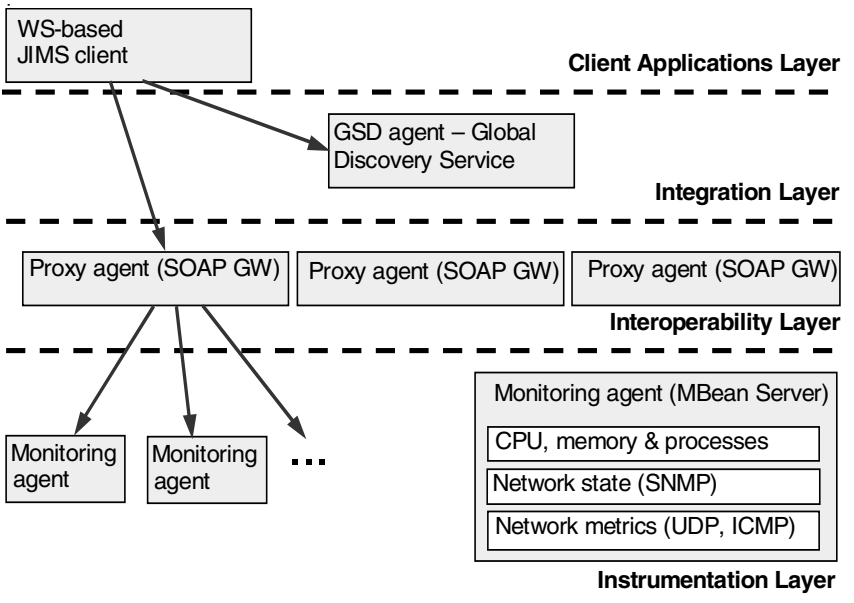


Fig. 1. JIMS architecture

Agents running in CE are called SOAP gateways and act as proxy agents between clients and WNs [9]. SOAP gateways discover and keep lists of all WNs in particular clusters and provide translation between interoperable WS/SOAP protocol to Java/RMI used by WNs.

Third layer, the integration layer, consists of certain number (typically three) of chosen SOAP gateways performing the role of Global Discovery Service. The fact of being GDS node is written in configuration file for CE agents in all clusters. As it was pointed out at the beginning, GDS was developed as a module for existing JMX-based monitoring system. There are no contraindications to have the same GDS module in all CE in the grid. Furthermore, all CEs must have the GDS module in order to perform registration in a set of well-known GDS nodes. Though not all CE agents are the GDS nodes, all keep the list of GDS nodes and all can keep the list of registered SOAP gateways, despite the fact, that only the lists in GDS nodes will be used. Detailed operation of integration layer will be described further in this chapter.

In described system there was chosen the number of three GDS nodes, which seems to be optimal both from the point of view of the network load generated by GDS nodes and the simplicity of making a decision of which services are really running and which are not, basing on the rule of majority. The number of GDS nodes results from the number of GDS hosts entries in Java configuration file (gds.properties), which is read during GDS startup. Nevertheless, all entries concerning the GDS nodes can be fully edited after the system startup, i.e. changed, added and removed.

The last layer, client application layer, is the layer where there is performed final processing of monitored data. This processing can concern visualization, performance prediction and benchmarking aspects. Description of this layer is not within the scope of the article; more information can be found in [11].

Though GDS existence is highly demanded, it's only an additional module for monitoring system, what means that it is not required for normal monitoring system operation. GDS is rather the utility providing the current view of available clusters, assuming that every cluster has JIMS monitoring system installed and working. It doesn't solve the problem of existing but not listed clusters due to the crash of the particular monitoring systems. However, successful GDS operation in production grid environment proves the stability and durability of presented solution.

3 WS-GDS Sequence Diagrams

Typical use case of JIMS monitoring system is shown in Fig. 2. In this scenario, client connects directly to Computing Element with running SOAP gateway and obtains the list of all available monitored Worker Nodes. Having the list of WNs, it requests certain parameters from monitoring agents operating in these WNs. This scenario assumes that client knows the address of SOAP gateway where there are required monitored stations. In order to gather information about all available WNs in all clusters, client has to maintain the list of all SOAP gateways in grid network, what is the main task of GDS.

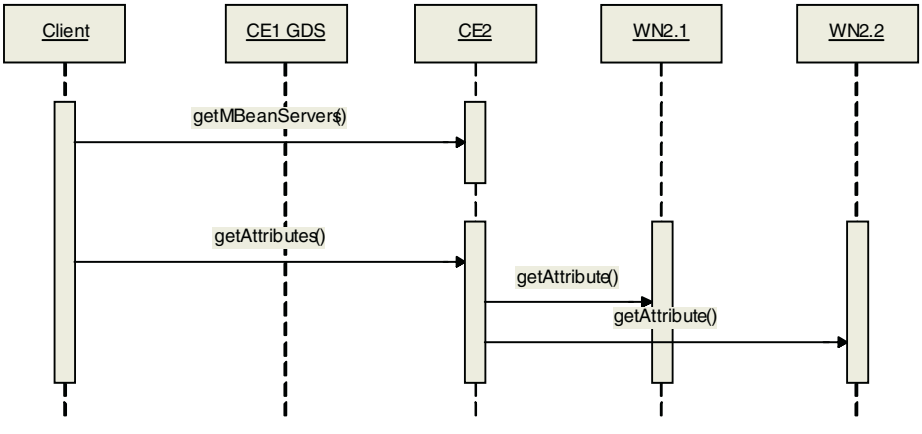


Fig. 2. Direct connection to CE

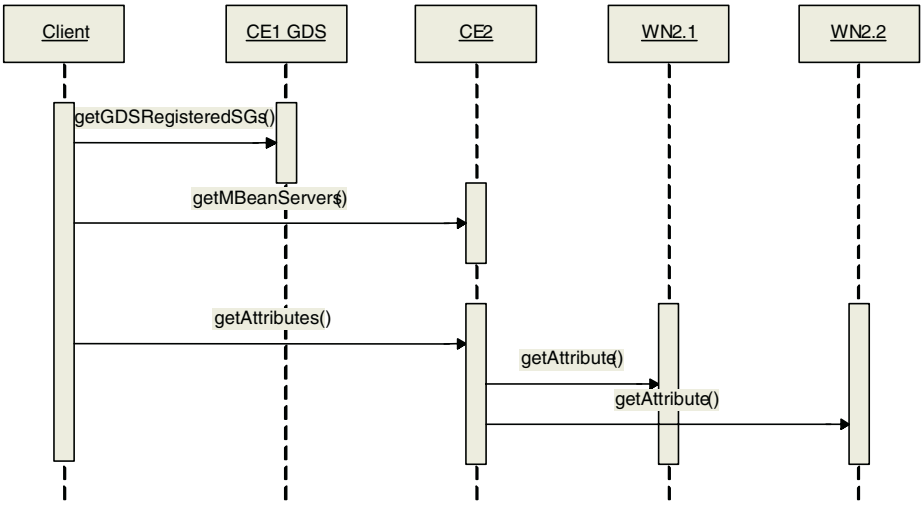


Fig. 3. CE localization using GDS node

There are the following features that the Global Discovery Service should provide:

- SOAP gateways registration facility and access to the list of currently registered SOAP gateways,
- heartbeat mechanism for removing old entries of not working SOAP gateways,
- timestamps when the registration took place,
- fail-over facility - GDS should use redundant GDS nodes for immunity to software and network failures,
- addresses of GDS nodes in all SOAP Gateways.

Usage scenario using GDS is shown in Fig. 3. It is similar to scenario shown in Fig. 2 and is supplemented by *getGDSRegisteredSoapGateways()* method, returning the list of valid SOAP gateway addresses, accessible at the time when the method was called. After this method call, communication between JIMS client and monitoring services takes place in the same way as before, i.e. client connects to SOAP gateway, requests monitoring parameters from WNs or executes a method on WNs to perform some measurements.

Important thing in GDS is the known set of certain number of SOAP gateways serving as GDS registries. These GDS nodes addresses should be configured in all SOAP gateways, because they should register themselves in GDS nodes. The list is exposed as an attribute of managed bean in GDS JMX module and is available for

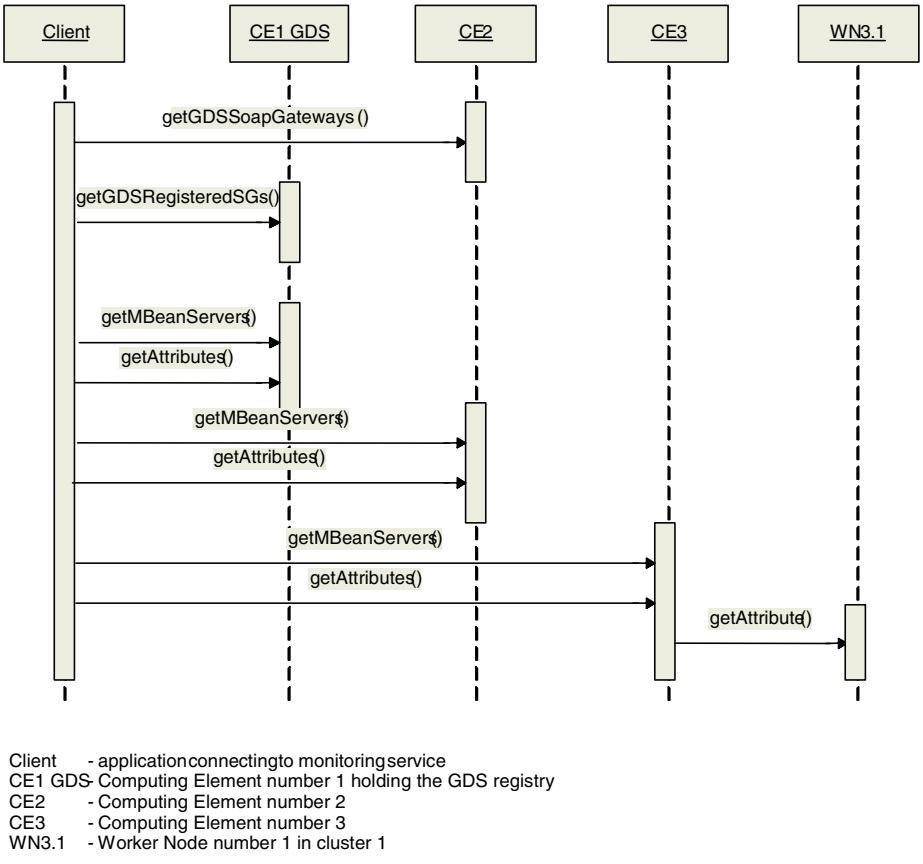


Fig. 4. GDS localization using CE with GDS MBean installed (in CE2)

clients at any time. Thanks to this, clients don't even need to keep the list of all GDS nodes, because they can maintain only the address of one SOAP gateway, not necessary the SOAP gateway performing the GDS role. It is only required that the GDS MBean module is installed and addresses of the GDS nodes are available, so the client can read them, connect to GDS node and read the full list of all available SOAP gateways.

Client can also connect to all available GDS nodes to choose the most reliable set of SOAP gateways using the majority rule. Having the list of SOAP gateways, it can query all sites with proxy agents and read information about all monitored nodes, what makes presented system the true grid utility, integrating all cluster monitoring systems and providing coherent view of all sites.

Described scenario is presented in Fig. 4. In this case there are three SOAP gateways installed in CE 1, 2 and 3 respectively. CE 1 is the chosen SOAP gateway performing also the role of GDS node, what is specified in configuration file. Let us assume, that the client does not know the addresses of GDS nodes, so chooses the first known address of CE (in this case it is the CE 2), which can be the Computing Element installed locally, in place where the client is running. From CE 2 the client

obtains the list of GDS nodes using *getGDSSoapGateways()* method. Having the list, the client connects to one of GDS nodes (in the picture it is CE 1) and executes the *getGDSRegisteredSoapGateways()* method (in the picture called *getGDSRegisteredSGs()*). In next step, the client connects to CE 1, CE 2 and CE 3, and requests information about interesting worker nodes, what was shown for CE 3. In order to get full list of WNs available through given SOAP gateway, the client connects to it and performs the *getMBeanServers()* method, which returns the RMI addresses of all WNs in cluster. These addresses contain the IP address and the serialized version of Java stub object, which can be useful for clients connecting directly to worker node. Next calls deliver only the values of requested parameters, though it is possible to indirectly call the particular WN's parameter, using the described RMI address, what is described in [720].

4 GDS Implementation Using Web Services and JMX

Presented system was implemented and deployed in European grid installation under CrossGrid project. The implementation is based on Java version 1.4, Java AXIS version 1.2 alpha and JMX - JSR 160. As many other JIMS modules, GDS was implemented as JMX module (MBean) running within SOAP gateway agent running on CE host. The GDS module implementation as MBean results in its high reusability. Furthermore, such design doesn't narrow its functionality in any aspect. Having it compiled and packaged as standard Java library (JAR – Java archive), it's possible not only to deploy GDS in any MBeanServer in other systems requiring global discovery functionality, but in any Java application as well, since MBean is a standard Java class with special interface exposing some attributes and methods for further usage. These attributes and methods are also exposed by the web interface using JMX HTTP Adaptor component, what is depicted in Fig. 5, 6 and 7.

In order to achieve the ability to deploy modules while the system is running, JIMS was equipped with standard JMX MBean for dynamic modules uploading and starting: the MLet MBean, available in JMX. In order to dynamically load GDS module to SOAP gateway, the GDS JAR (Java Archive) file should be first installed on a HTTP server. Secondly, there should be prepared a mlet file containing description of provided JAR file and it's content: class, module name, etc. In the end, in MBeanServer there should be invoked the *getMBeansFromURL* method with MLet URL as a parameter. Loaded module is shown in Fig. 5. There are three key components exposed in this module:

- GDSRegisteredSoapGateways – Java String array containing all SOAP gateways registered in GDS; it should contain the full list of all JIMS monitoring systems running GDS, installed in the grid (Fig. 7),
- GDSSoapGateways – Java String array containing all SOAP gateways performing the role of GDS nodes. Typically, it contains three entries of SOAP gateways that maintain the lists of registered SOAP gateways (Fig. 7),
- LocalSoapGateway – the URL of local SOAP gateway, which is periodically registered in GDS nodes (Fig. 5). For development purposes the time between two registration was set to 30 seconds, but as well as the number of GDS nodes and their addresses, it is fully configurable through the GDS configuration file.

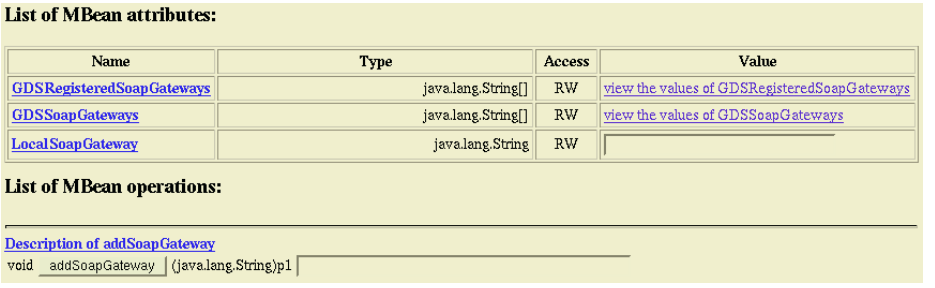


Fig. 5. GDS MBean attributes and methods

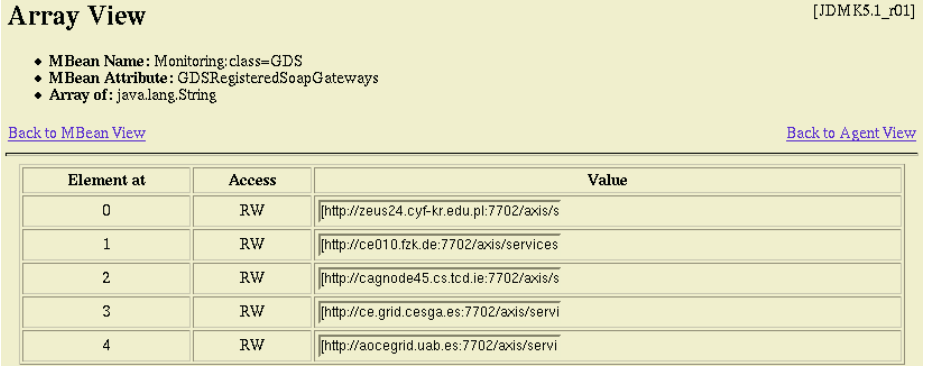


Fig. 6. GDS SOAP gateway registry

Fig. 7. GDS nodes list

5 GDS Evaluation and Related Projects

The most important thing in presented solution is its simple implementation based on Web Services providing connectivity between SOAP gateways in different clusters. Using WS along JMX allowed rapid development and implementation of quite complicated system. Using existing WS infrastructure and exposing monitored

parameters and methods for network measurements, it was easy to prepare another module running within CE and to allow the clients to invoke the dedicated method *addSoapGateway()*. Next, the client can get addresses of all GDS nodes and obtain the complete list of running JIMS SOAP gateways. Such solution could be deployed in existing monitoring infrastructure without any configuration changes what is big advantage and speeded up deployment and test of the system administrated by many autonomous systems in wide area network.

In the future performance evaluation of the presented system should be carried out. The most suitable metric seems to be the time of convergence. This metric can be used also for cluster level discovery service evaluation, described in [9]. Such convergence time will strongly depend on GDS update time, which is 30 seconds by default. It assures fast SOAP gateway list convergence, but requires more CPU power on CE elements. In stable environments, there should be considered definitely longer update interval, saving expensive CPU time spent on extensive XML processing due to the use of WS and JMX.

Discovery service should be also equipped with heartbeat mechanism for dynamic SOAP gateways removal while they are inaccessible during given period of time [3]. Proposed solution can be the same as it is used in RIP routing protocol, marking SOAP gateways entries as invalid after certain period of time (typically triple times longer than the period of GDS updates) and then removing them after another well defined period of time.

There is also place for optimizations in implementation area. While the system was being developed, Sun Microsystems issued an upgraded version of Java VM version 5.0, equipped with support for JMX. It could be interesting to adapt the system to this new version of Java, what should result in further implementation simplifications, because there should be no need to use additional packages for Java Management Extensions.

To sum up, WS-based Global Discovery Services features: simplicity (only one class to implement; GDS and GDSMBean as an interface), reliability (immunity to failures of up to $(n-1)$ nodes in the collection of n nodes acting as a GDS registry nodes), non-intrusiveness to actual configuration of firewalls and the whole monitoring system resulting in fast deployment time and possible integration in other systems based on Java and JMX. Its simple concept can be easily implemented using other languages and technologies. Nevertheless, presented system proves that Java-based WS and JMX provide sufficient level of abstraction and robustness to implement rich functionality using very little effort and implementation time, just using services already provided by JIMS and JMX.

6 Conclusions

JMX, Dynamic Loading service and Web Services allow implementation of functionally complicated systems in a simple way. It should be emphasized that the whole GDS system was implemented in a few days and was installed without any changes to existing system, including firewalls, packages and other configuration.

It requires only any new monitoring system to be equipped with the new service in order to register itself in the GDS registry nodes.

Presented approach was successfully tested in the grid network monitoring system consisting of 16 clusters, providing fast convergence and easy integration with existing infrastructure, without any notice to network and firewall administrators. It's also important, that this solution has the two key capabilities:

- automation of deployment,
- remote deployment,

which are lacking in other considered solutions, based on JXTA or UDDI. Current implementations of JXTA platforms cannot be used in environments with requirement for full automatic installation and configuration. The UDDI solution for Web Services is also too complicated, requiring installation of separate database and even more nontrivial configuration.

The last thing that needs discussion is security. Besides the functional features, monitoring system should provide firm security mechanism. Current version of monitoring system doesn't support authentication facility, that's why in the future there should be developed security mechanism to prevent unauthorized access to monitored parameters. Such mechanism could probably use standard solution such as GSI (Grid Security Infrastructure) and X.509 certificates [12].

References

1. Sun Microsystems: Java Management Extension Reference Implementation (JMX), http://java.sun.com/products/JavaManagement/
2. Sun Microsystems: JMX Remote API Specification (JMX Remote API), http://developer.java.sun.com/developer/earlyAccess/jmx/
3. Sun Microsystems: Java Dynamic Management Kit Specification v. 4.2 (JDMK), http://java.sun.com/products/jdmk/, JDMK42_Tutorial.pdf, pp. 121-128
4. Sun Microsystems: Grid Engine, http://gridengine.sunsource.net/
5. W3C: Web Services Activity, http://www.w3.org/2002/ws/
6. The Open Grid Services Infrastructure Working Group (OGSI-WG): OGSI Specification, http://www.ggf.org/ogsi-wg
7. The Open Grid Services Architecture Working Group (OGSA-WG): Globus Tutorial, www.globus.org/ogsa/
8. J. Midura, K. Balos, K. Zielinski: Global Discovery Service for JMX Architecture, ICCS 2004, LNCS 3038, pp. 114-118, 2004
9. K. Balos, L. Bizon, M. Rozenau, K. Zielinski: Interoperability Architecture for Grid Networks Monitoring Systems, CGW '03, Workshop Proceedings, pp. 245-253, 2004
10. K. Balos, K. Zielinski: JMX-based Grid Management Services, Workshop on Networks for Grid Applications - GridNets Proceedings, 2004
11. K. Balos, D. Radziszowski, P. Rzepa, K. Zielinski and S. Zielinski: Monitoring Grid Resources: JMX in Action, Task Quarterly 8 No 4, pp. 487 – 501, 2004
12. The Globus Alliance: Grid Security Infrastructure, http://www-unix.globus.org/toolkit/docs/3.2/gsi/key/index.html

Rapid Distribution of Tasks on a Commodity Grid

Ladislau Bölöni[1], Damla Turgut[1], Taskin Kocak[1], Yongchang Ji[2], and Dan C. Marinescu[2]

[1] Department of Electrical and Computer Engineering
[2] School of Computer Science,
University of Central Florida,
Orlando, FL 32816
{lboloni, turgut, tkocak}@cpe.ucf.edu,{yji,dcm}@cs.ucf.edu

Abstract. The global internet is rich in commodity resources but scarce in specialized resources. We argue that a grid framework can achieve better performance if it separates the management of commodity tasks from the tasks requiring specialized resources. We show that the performance of task execution on a commodity grid is the delay of entering into execution. This effectively transforms the resource allocation problem into a routing problem.

We present an approach in which commodity tasks are distributed to the computation service providers by the use of a forwarding mesh based on randomized Hamilton cycles. We provide stochastically weighted algorithms for forwarding. Mathematical analysis and extensive simulations demonstrate that the approach is scalable and provides efficient task allocation on networks loaded up to 95% of their capacity.

1 Introduction

The computational grid (and the internet at large) is rich in commodity resources but scarce in specialized resources. There is a large number of PC class hardware (Windows and Apple desktops, Unix and Linux workstations) with typically very low resource utilization. On the other hand, there is a scarcity of specialized resources, such as supercomputers, vector processors, specialized input and output devices and so on. Typically, the need for specialized resources is dictated by the nature of the application and, less often, by the chosen implementation.

If we look at the state of the art for distributed high performance computing, we see two different approaches:

– The computational grid community develops software which manages scarce specialized resources. Although the vision of grid computing was refined several times ([4] → [6] → [5] → [2]) the main deployment of grid applications are for projects with expensive specialized hardware. Examples of testbeds are the grid projects of the National Partnership for Advanced Computational Infrastructure (NPACI) and National Computational Science Alliance

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 721–730, 2005.

(NCSA) in the US or the European DataGrid project. The grid computing projects developed at IBM, Sun and Hewlett Packard are also largely fall in this category.
- A number of distributed computing initiatives are exploiting the abundance of commodity resources for solving highly parallelizable applications. Examples are SETI@Home [16], Folding@Home [13], the cryptographic challenges sponsored by RSA laboratories [15] or the Mersenne prime search [14]. The Berkeley Open Infrastructure for Network Computing (BOINC, [1, 12]) proposes to provide a framework more general than the SETI@Home project, which can be shared by a number of projects following this pattern of interaction. These projects, which rely on donated processor time are sometimes referred as "public computing".

Both approaches target grand challenge applications. The applications targeted by the grid computing community however, are more general than the typical public computing approaches. On the other hand, SETI@Home and the related applications are highly successful in harnessing large amount of cheap computing resources.

We note that many high performance computing workflows contain both specialized and commodity tasks. For the specialized tasks, the best thing the workflow engine can do is to queue them at the appropriate specialized providers, for instance through a system such as Condor [11]. For commodity subtasks however, this approach is not appropriate. There are a very large number of community service providers (on the order of millions), which makes it difficult to deploy any kind of centralized distribution system.

We note that if a task is executed on a commodity hardware, the main determining factor of the termination time is the time at which the task is taken into execution. Furthermore, given the abundance of the commodity resources, it is likely that if a task needs to be queued at a certain host, it is almost sure that somewhere on the internet there is a task which can take it into execution immediately. Under this assumption, the task allocation problem is reduced to a specialized routing problem. A similar idea is proposed in [7, 3]. The Wire Speed Grid Project at the University of Chalmers [17], proposes an architecture in which the task allocation is performed in a hardware accelerated manner on the network routers. As our tasks have a relatively long execution time, an application layer implementation would provide the same benefits.

2 Commodity Components in Grand Challenge Applications

Grand challenge applications range from the application of relatively simple algorithms on massive amounts of data (such as the SETI@Home project), to exhaustive search of a complex combinatorial problems with small amounts of input and output data (e.g. cryptographic analysis). Many of the high performance applications however, are what we call *grid workflows*. Problems with

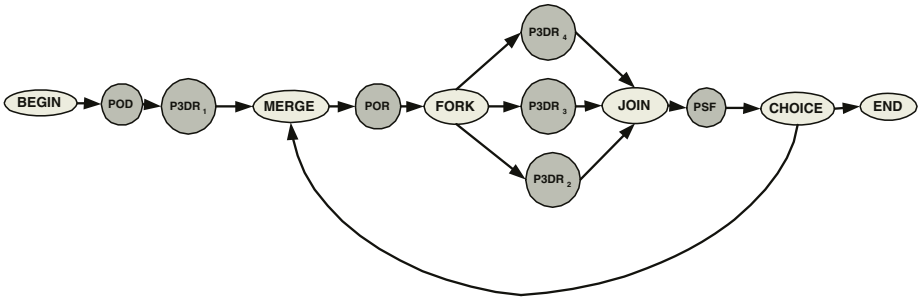


Fig. 1. Workflow for 3D virus structure reconstruction based on 2D electronmicroscope data

significant scientific and commercial interest such as predicting trajectories of hurricanes (WRF, ROMS), virus structure reconstruction, protein folding, DNA sequencing for individuals or designing custom drugs fall in this category.

Grid workflows involve a sequence of steps, such as data collection, filtering, computation, modelling and visualization. They frequently require interaction with the user in form of computation steering. The process can involve testing alternative hypothesis, thus the execution sequence can vary between individual runs. These problems are usually described as a workflow model of directed acyclic graphs, although cycles are sometimes necessary. The nodes of the graph are subtasks with different resource requirements.

Case Study: Structural Virology Application. In the following, we describe a typical grid application from the field of structural virology with which the authors have extensive experience. The 3D atomic structure determination of macromolecules based upon electron microscopy [9, 10, 8] consists of the following steps:

1. Extract individual particle projections from micrographs and identify the center of each projection.
2. Determine the orientation of each projection.
3. Carry out the 3D reconstruction of the electron density of the macromolecule.
4. Dock an atomic model into the 3D density map.

Steps 2 and 3 are executed iteratively until the 3D electron density map cannot be further improved at a given resolution; then the resolution is increased gradually. The number of iterations for these steps is in the range of hundreds and one cycle of iteration for a medium size virus may take several days. Typically it takes months to obtain a high resolution electron density map. Then Step 4 of the process can be pursued. Once we have a detailed electron density map of the virus structure, we can proceed to atomic level modelling, namely placing of groups of atoms, secondary, tertiary, or quaternary structures on the electron density maps.

The grid workflow for this procedure is described in Figure 1. The experimental data is collected using an electron microscope, and the initial input data

is 2D virus projections extracted from the micrographs. The goal of the computation is to construct a 3D model of the virus at specified resolution or the finest resolution possible given the physical limitations of the experimental instrumentation. First, we determine the initial orientation of individual views using an "ab initio" orientation determination program called `POD`. Then, we construct an initial 3D density model using our parallel 3D reconstruction program called `P3DR`. Next, we execute an iterative computation consisting of multi-resolution orientation refinement followed by 3D reconstruction. The program for orientation refinement is called `POR`.

In order to determine the resolution, the input data is split in a number of streams. For each stream, we construct a model of the electron density maps and determine the resolution by correlating the models with a program called `PSF`. The iterative process stops whenever no further improvement of the electron density map is noticeable or the goal which we specified is reached.

Although the grid workflow contains multiple data dependencies, it has components which can be executed on commodity hardware. The data acquisition step requires a computer connected to an electron microscope, and significant human work. The `POD` and `PSF` steps are an parallel programs, utilizing the message passing interface (MPI) and they require machines with very fast networking capabilities. The cheapest hardware which would still do the work is a Beowulf cluster of 32 or more computers and Gigabit Ethernet interconnects. The `P3DR` steps however are parallelizable as they will be correlated only in the next step of the workflow. Although computationally intensive, they can be run on commodity hardware. We need to note however, that components of the workflow requiring specialized hardware depend on results coming from commodity tasks.

3 System Architecture

The participants in the distributed task allocation algorithm are:

Application Client (AC). A host which desires to run a grid application, some part of which is expressible as task solvable by a commodity algorithm. The application client is usually controlled by a human user.

Commodity Resource Providers (CRP). Computers which can run one or more of the algorithms in the commodity algorithm server.

Distribution Nodes (DN). Computers which are able to forward tasks according to the distribution policy. All the CRPs are also distribution nodes, but a grid deployment might introduce distribution nodes to help the distribution of the packets. The application client needs to be in contact with at least one distribution node, which represents the entry point into the network.

Commodity Algorithm Server (CAS). A file service system which provides the standard implementation algorithms. This is normally a simple FTP or HTTP based service with a specific naming convention.

Commodity Solution Checker (CSC). A trusted web service which given a canonic description of a task and a proposed solution checks if the proposed

solution is an acceptable solution of the task. The CSC enables us to use CRPs with lower levels of trust. For some algorithms this check implies independent execution of the algorithm and the comparisor of the results. For many algorithms, the result can be verified without repeated execution.

The general process of the algorithm is as follows:

(1) The AC formulates a commodity problem as a task packet and sends it to one of its entry points.
(2) When a packet reaches a distribution node which is also a CRP, it is either bidded for its execution, or distributed/forwarded according to a *distribution policy*. The bidding is sent to the AC and a preliminary allocation is done as a soft state. The reply deadline is specified in the bid, and it is a relatively short period of time (at most several minutes).
(3) The AC sends a task award packet, containing the descriptions of the access methods of the application input. This might be contained in the confirmation packet itself or it can be a remote reference, accesible by protocols such as GridFTP. The bid might contain some setup information, such as whether the bidder needs to download the required algorithm fro the CAS or it has it in its local cache.
(4) The CRP starts the task execution process and sends a TASK_STARTED confirmation packet.
(5) [Optional] Algorithm download. If the provider does not have the required algorithm installed, it can download it from a trusted algorithm provider.
(6) [Optional] Data preparation. The input data of the process is loaded by the application using the GridFTP protocol. If the application input is very small, it can be sent in the task award packet.
(7) The CRP executes the algorithm on the specified data. It uploads the results to the locations indicated in the task specification packet. In case of success it reports to the application with a TASK_TERMINATED Packet. The CRP then becomes available for processing other tasks.

4 The n-Cycle Task Distribution Algorithm

The goal of the task distribution algorithm is to deliver tasks to commodity resource providers. With the number of CRPs involved (on the order of millions), scalability is of utmost importance. Having millions of hosts changing their availability on a minute-per-minute basis centralized algorithms based on global information are not appropriate.

The n-Cycle algorithm we propose uses only limited local information, it is virtually indefinitely scalable and performs efficient task distribution for grids loaded as high as 95% of their nominal capacity. The algorithm can be divided in two parts: the creation and maintenance of the forwarding mesh and the forwarding algorithm.

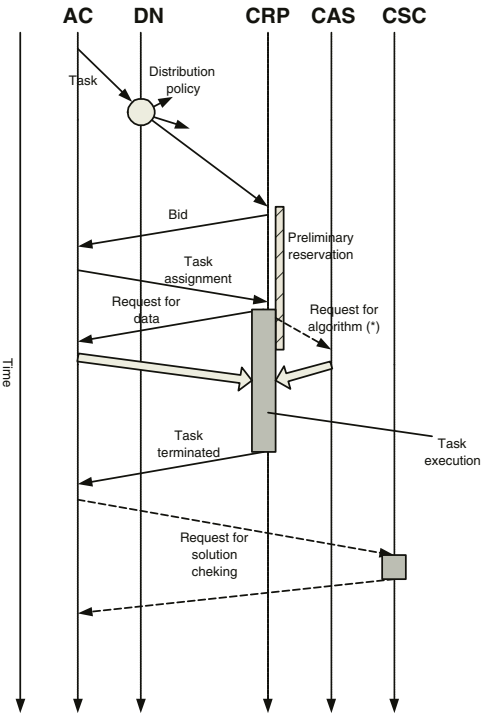


Fig. 2. The flow of the task allocation process

4.1 Creation and Maintenance of the Forwarding Mesh

The n-Cycle algorithm creates a forwarding mesh comprised of directional links. For any link $A \rightarrow B$, we will have task forwarded from A to B and status information propagated from B to A. The links of the forwarding mesh form n separate Hamiltonian cycles connecting all the elements in the grid node. The cycles are formed randomly, we are not interested in optimizing the length of the cycle. The randomness of the cycles is an important part of the algorithm. Figure 3 shows a 3-Cycle forwarding mesh on a grid of 5 nodes. For any n-Cycle mesh, every individual node will have n nodes "upstream" and n nodes "downstream" from it. The node forwards tasks to the upstream nodes and receives status updates from them. Similarly, the node receives tasks from the downstream nodes and forwards status updates to them.

4.2 Distributing Tasks on the Forwarding Mesh

One of the remarkable properties of the n-Cycle forwarding mesh is that a significant majority of the nodes can be reached by only $log_n(|W|)$ hops.

We can design a *random wandering* task allocation algorithm, with the following rule: if current host is free, take the incoming task into execution. If not,

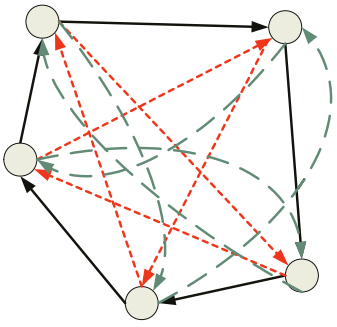


Fig. 3. A 3-Cycle forwarding mesh on a network of 5 nodes

then forward randomly to one of the uplink nodes. As we showed before, we are interested in bringing the task into execution as quickly as possible, which means that we need to minimize the number of hops.

For a random wandering algorithm, the number of hops depends on the average load of the network p. In a first approximation, for any number of hops h, the probability that a node will be allocated in less than h hops is $(1-n)^h$. Although this approach leads to satisfactory average values as long as the load is not getting close to 100%, the maximum values can be (potentially) indefinitely long. The advantage of a random wandering algorithm is that it operates without any information about the state of the network.

In the following we introduce a *weighted stochastic algorithm* which uses information collected from upstream nodes in the forwarding decision. In our simulation studies, we show that this algorithm leads to significantly better performance with an acceptable cost. Every node maintains its weight w which intuitely represents the desireability of the node as a forwarding target for a task. The weight w is composed in equal parts from (a) the ability of the node to receive a task for execution (b) the weights of the nodes downstream from the node. The weight w is propagated to the upstream nodes. A change in the weight is propagated only if it exceeds a threshold δ, preventing floods of updates.

At any given node, a task is either taken into execution (if the node is free), or forwarded to one of the upstream nodes with a probability proportional with their weight (as seen by the current node). The complete algorithm is presented in Algorithm 1.

5 Simulation

We have used the YAES [18] simulation framework to simulate the behavior of the algorithm. Table 1 illustrates the input and output parameters of the simulation as specified in the YAES configuration files.

Algorithm 1. Weighted stochastic task forwarding

When task T received by node N
  If STATUS == free
    take T into execution
    STATUS == busy
  Else
    forward to upstream node i with probability $\frac{w_i}{\sum_{k=1,n} w_k}$
  calculate new weight $w_{new} = f(STATUS, w_i)$
  If $|w - w_{new} < \delta|$
    send the new weight to all upstream nodes
    $w = w_{new}$
When weight w received from i-th downstream node
  $w_i = w$
  calculate new weight $w_{new} = f(STATUS, w_i)$
  If $|w - w_{new} < \delta|$
    send the new weight to all upstream nodes
    $w = w_{new}$

Table 1. YAES simulation parameters

Input parameters	
Number of grid nodes	100,000
Forwarding mesh	5-Cycle
Task arrival	Poisson-distributed arrival, mean 10...200 tasks/sec
Task servicing	Normally distributed, mean 60 sec/task
Simulation time	5000 seconds
Output parameters (Measurements)	
Hops per task	Number of hops a task is forwarded until it finds a host for execution (avg, max)
Average load	Ratio of busy vs. total nodes
Discarded tasks	Number of tasks which were discarded

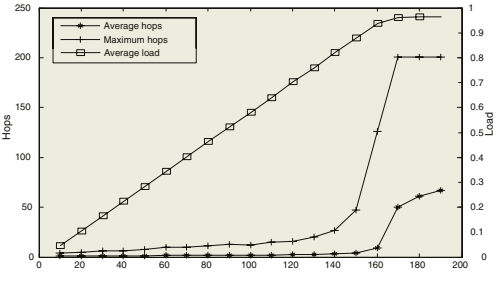


Fig. 4. Average number of hops, maximum number of hops and network load using weighted stochastic forwarding on the n-Cycle forwarding mesh

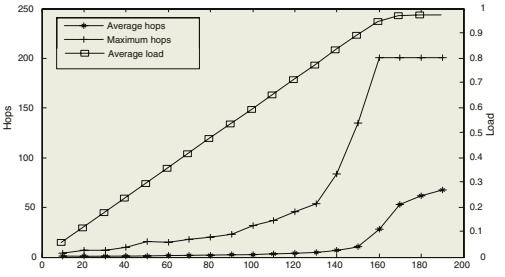


Fig. 5. Average number of hops, maximum number of hops and network load using random forwarding on the n-Cycle forwarding mesh

Figure 4 presents the average and maximum number of hops it takes for a task to be allocated and the total network load in function of the average number of arriving tasks. We note that both the average and maximum number of hops is staying virtually constant at a very low number, up to loads approaching 95%. At that moment the number of hops increases dramatically as the algorithm struggles to find free nodes in an overwhelmingly busy network.

The relatively constant number of nodes for moderate loads is explained by the single insertion point. The nodes closer to the insertion point will be filled in relatively quickly, so the majority of tasks need to "hop over" the busy nodes in this area. A good approximation of the size of this constant value is $log_N(n)$ which in our case is $log_5(10000)$, approximately 5.7. If we choose a random insertion point, we will obtain a diagram with a similar shape, but with an average number of hops for lightly loaded networks much smaller (about 1-2 hops).

In a different simulation run, Figure 5 presents the random walking algorithm. For small loads, this algorithm also shows very good results (due to the randomizing nature of the N-Cycle mesh). However, for greater loads, the maximum number of hops start to increase. For instance, at load of 90% the maximum will be as high as 100 hops vs. about 20 hops for the stochastically weighted algorithm.

6 Conclusions and Future Work

In this paper, we introduced an algorithm for allocating commodity tasks on a computational grid. Our analysis and simulation studies show that (a) the algorithm is scalable (b) it proved to be very efficient in allocating tasks to free computational service providers.

Our future work includes more extensive mathematical analysis of the algorithms. It is of special interest on modeling the influence of the estimation of the w_i values of the upstream nodes, as higher accuracies for these values require higher message traffic on the mesh. We also plan to extend the algorithm to heterogeneous networks.

References

1. D. P. Anderson. Public computing: Reconnecting people to science. In *Proceedings of the Conference on Shared Knowledge and the Web*, Nov 2003.
2. M. Baker. Ian Foster on Recent Changes in the Grid Community. URL `http://dsonline.computer.org/0402/d/o2004a.htm`.
3. B.Liljeqvist and L.Bengtsson. Grid computing distribution using network processors. In *Proc. of the 14th IASTED Parallel and Distributed Computing Conference*, Nov 2002.
4. I. Foster and C. Kesselman, editors. *The Computational Grid: Blueprint to a New Computer Infrastructure*. Morgan-Kauffman, 1998.
5. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An open grid services architecture for distributed systems integration. URL `http://www.globus.org/research/papers/ogsa.pdf`.
6. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.
7. A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grif environments. In *Proceedings of the International Workshop on Grid Computing, Denver, CO, November 2001*, 2001.
8. Y. Ji, D. C. Marinescu, W. Zhang, and T. S. Baker. Orientation refinement of virus structures with unknown symmetry. In *Proceedings of the 17th Ann. Int'l Parallel and Distrib. Processing Symposium Nice, France*. IEEE Press, 2003.
9. D. C. Marinescu and Y. Ji. A computational framework for the 3d structure determination of viruses with unknown symmetry. *Journal of Parallel and Distributed Computing*, 63(7-8):738–758, 2003.
10. D. C. Marinescu and Y. Ji. A computational framework for the `3d` structure determination of viruses with unknown symmetry. *Journal of Parallel and Distributed Computing*, 63:738–758, 2003.
11. D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.
12. Berkeley Open Infrastructure for Network Computing. URL `http://boinc.berkeley.edu/`.
13. Folding@Home project. URL `http://www.stanford.edu/group/pandegroup/folding/`.
14. Mersenne Prime search. URL `http://www.mersenne.org/prime.htm`.
15. RSA Challenge. URL `http://www.rsasecurity.com/rsalabs/challenges/`.
16. SETI@Home project. URL `http://setiathome.ssl.berkeley.edu/`.
17. The Wire Speed Grid project. URL `http://www.ce.chalmers.se/staff/labe/Wire Speed Grid Project.htm`.
18. YAES: Yet Another Extensible Simulator. URL `http://netmoc.cpe.ucf.edu/Yaes/Yaes.html`.

Modeling Execution Time of Selected Computation and Communication Kernels on Grids

M. Boullón[1], J.C. Cabaleiro[1], R. Doallo[2], P. González[2], D.R. Martínez[1], M. Martín[2], J.C. Mouriño[2], T.F. Pena[1], and F.F. Rivera[1]

[1] Dept. Electronics and Computing, Univ. Santiago de Compostela, Spain
[2] Dept. Electronics and Systems, Univ. A Coruña, Spain

Abstract. This paper introduces a methodology to model the execution time of several computation and communication routines developed in the frame of the CrossGrid project. The purpose of the methodology is to provide performance information about some selected computational kernels when they are executed in a grid. The models are based on analytical expressions obtained from exhaustive monitorized measurements. Even though the kernels that are considered in this work include both applications dependent and general purpose, the methodology can be applied to any kind of kernel in which the most relevant part in terms of execution time is due to computations and/or communications. We focused on MPI-based communications. In addition, an interactive Graphical User Interface was developed to summarize and show the information provided by the models from different views.

1 Introduction

Performance evaluation, instrumentation, prediction and visualization of parallel codes has been found to be a complex multidimensional problem [1] in parallel and distributed systems. This situation is critical in grid environments. Tuning the performance of codes on distributed memory systems has been a high time-consuming task for users. When programming these systems, the reasons for poor performance of parallel message-passing and data parallel codes can be varied and complex, and the users need to be able to understand and correct performance problems in order to achieve good results. This is especially relevant when high level libraries and programming languages are used to implement parallel codes. Performance data collection, analysis, prediction and visualization environments are needed to detect the effects of architectural and system variations.

In high-performance computing, applications performance is very sensitive to problem features such as code and data partitioning, and machine computation and communication parameters. Performance prediction is an important engineering tool that provides timely feedback on design choices in program synthesis as well as in machine architecture development. Apart from prediction accuracy,

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 731–740, 2005.

prediction cost largely determines the utility of the tool. Performance prediction approaches take many shapes, the choice of underlying modeling formalism depending on the desired trade off between prediction cost and accuracy. Although potentially accurate, modeling formalisms such as stochastic Petri nets [2] or process algebras [3] are not attractive due to the exponential solution cost. Although approaches based on combinations of directed acyclic task graphs and queuing networks [4, 5] pair comparably high modeling power with a high efficiency, the polynomial time complexity of the solution process still entails considerable cost for very large problem sizes. In analytical approaches, the application is transformed into an explicit, algebraic performance expression. In contrast to the above numeric approaches, symbolic prediction techniques offer even lower solution cost that is less dependent on problem size. Manual approaches, such as BSP [6] and LogP [7], modeling cost is significant because the labor-intensive and error-prone derivation effort. Alternatively, symbolic prediction techniques based on stochastic direct acyclic graphs and deterministic direct acyclic graphs [8, 9] offer a mechanized scheme. However, unacceptable prediction errors arise when performance is largely dominated by contention for resources such as locks, servers, processors, networks links, or memories.

Sophisticated performance prediction tools are being developed by a number of groups. In particular we can cite the PACE toolset [10], PERFORM and LEBEP[11], DIMEMAS [12], INSPIRE [13], Carnival [14], ALPES [15], P3T [16], PerFore [17] and Bricks [18]. Other works used prediction as part of a software-aided approach for particular applications, e.g. solvers - we cite SPEED [19], SciRUN [20] or PARAISO [21]. A long-term project is the development of the AlpStone [22] project, that simulates parallel performance using information from a database of kernel benchmarks and underlying hardware parameters using a "skeleton application" approach. The kernel benchmarks are representative of "algorithm classes" and a "synthetic program" is built from these to represent the real application.

The application of these techniques has so far been largely restricted to accurate prediction of performance of code kernels with the goal of automatic parallelization or execution steering. And all of them focus on specific parallel systems, at most on clusters of workstations, like DIMEMAS or Carnival, but to our knowledge, no one is currently fully adapted to grid environments except Bricks and DAMIEN [23] that is a project to do that for DIMEMAS. On the other hand, most of these tools require simulation of the codes, that takes much time, also they are not application-dependent, and some of them are for specific programming platforms, P3T is for High Performance Fortran programs, AlpStone and Carnival are for PVM programs, PERFORM is just for sequential systems, and PerFore is integrated in a specific compiler.

In this work we establish a methodology to obtain analytical models based on exhaustive measurements of execution times obtained in a monitorized environment. The idea is to correlate execution times with monitorized information. In particular we consider computational power of each node as well as network bandwidth and latency.

The proposed models are application-dependent, and they focus on some kernels from the applications of the CrossGrid project [24] as well as other general purpose kernels. In this way, the results will be more accurate, and as the performance will be modeled by simple analytical expressions, it is very fast. A model for each MPI-coded kernel on the grid is established.

An additional and important feature is that the information is shown in a friendly interactive visual tool for users or developers of applications, This GUI uses the analytical models to establish performance predictions. In addition, this tool can include detailed information about specific parameters of the codes, as well as the predicted information about the execution of these codes. In fact, it should be an interface that provides interactivity in the analysis of the behavior of the codes under different conditions (number of processors, distributions, input data, network parameters, ...).

This GUI is devoted to be used by applications developers and users that are interested in analyzing the performance of the selected kernels under different scenarios. The results could be used to modify the parameters of the parallel execution of the application, like the number of nodes, the size of the problem, the distribution of data, etc. To enable a user to quickly find his way in the multidimensional design space, the GUI needs to be used interactively. In long term, it can be also used by resource brokers and schedulers to select the best platform according to the predicted figures offered by the tool.

Some of the main features of the tool are:

- It is an application dependent tool. It is specifically applied to selected kernels that were analyzed in terms of performance to obtain analytical models. Anyway, it can be also broadly used to study the performance of communication routines themselves.
- It is an interactive tool in the sense that the user can easily change the parameters that characterize the system or the problem, and then analyze their influence.
- It uses analytical models, and therefore the predictions are obtained fast. This feature is very important for the interactivity.
- It is specifically developed for heterogeneous systems.

In next sections, the methodology to obtain the models is described in detail.

2 A Methodology to Model Execution Times

The methodology is divided into two main stages. The first one is a study about the behaviour of the kernels that is useful to establish the parameters to be considered in the models. The second stage is to obtain the models by correlating execution times under different grid conditions and considering a range of values for the parameters that characterize the kernels.

2.1 Static Characterization of the Kernels

Initially, the kernels were characterized statically, in terms of the precise number of relevant events. One of these events is the number of floating point operations (FLOPs) required for the target kernel. This number were counted manually or using tools like PAPI [25] when it was necessary. After that, this value is summarized in simple algebraic expressions involving parameters of the kernel, such as the size of some matrices, or the number of iterations of some loop, or the number of non-zero entries in a sparse matrix, or the grade of some polynomial preconditioners, etc. All this parameters are statically established.

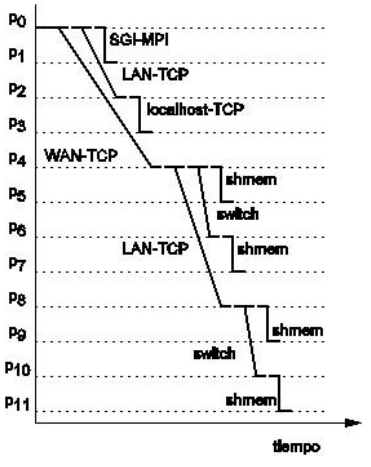


Fig. 1. A broadcast on 12 nodes

The study of the communication patterns generated by the MPICH-G2 routines used in the kernels is essential for predicting their overheads. In the same way as for the number of FLOPs, information about the number and size of the communications performed by each processor can be statically stated. In order to model MPI collective communications, the behaviour of many collective routines in terms of individual point to point communications were extracted. In this way, with the information about latency and bandwidth, the cost of these routines can be estimated.

For the communications kernels, MPICH-G2 distributes the processors in groups in different levels according to the communication behaviour. For example, level 0 is for WANs, level 1 is for LANs, and so on. Therefore, characterizing collective communications in MPICH-G2 in a set of point to point communications is based on the hierarchy of protocol levels: WAN, LAN, Local, ... In particular:

- The MPI_Bcast is implemented sequentially in level 0, and as a binary tree in other levels.
- MPI_Scatter is sequential in level 0, and a binary tree in other levels.

- MPI_Gather is also sequential in level 0, and a binary tree in other levels.
- The associative MPI_Reduce is implemented as a MPI_Bcast but in reverse order, and it includes the arithmetic or logic operation in each processor.
- The no associative MPI_Reduce is implemented as a MPI_Gather, and after that, the operation is performed sequentially in the root processor.
- The implementation of the MPI_Barrier is based on a hypercube communication in each level followed by an all-to-all communication in level 0.

Figure 1 shows an example of this hierarchical structure for a broadcast in a system with 12 nodes distributed in different levels.

2.2 The Models

To obtain a precise model for the performance of the computational kernels, a large number of executions of them under different scenarios were performed. In particular, the kernels were executed on different sites and on the whole grid.

As soon as all this static information about computations and communications was modelled, we deal with the development of the performance models. In this stage, JIMS [26], a monitoring tool developed in the CrossGrid project, was used. The cost of each kernel is measured through a large number of executions under different network features. Even thought JIMS offers a broad amount of information, just a reduced set of its functionalities is needed for our purposes. In particular:

- The online workload per CPU is used to establish the performance models for computations.
- The latency and bandwidth between each pair of processors to define the models for communications.

The main idea of our methodology is to obtain the correlation between monitoring information and features of the kernel with measured execution times. This method is based on the concept of "cube of tests". Consider, as an example, a point to point communication of a certain size. This kernel is executed K times in a short period of time, producing K measures of runtime named T_i. Consider that M monitoring tests were performed in this short period of time, producing M measures of latency and bandwidth named L_j and W_j respectively. A cube of tests is defined as the cube in a, in this particular case, three dimensional space $\{L, W, T\}$ limited by the minimum and maximum values achieved for these three parameters. Figure 2 illustrates a cube of tests. Note that this cube is defined for a certain size of the message. Therefore a fourth dimension has to be taken into account to obtain the model, that is, the size of the message.

Figure 3 illustrates how the monitorized information is extracted when the kernels are executed. In order to minimize the number of executions that are influenced by the monitoring process, their number have to be higher than the number of monitoring tests. According to our experience, about 10 executions of the kernel between two consecutive monitoring stamps are enough.

The monitoring information obtained in the period of time that defines the measurements must be homogeneous. If this is not the case, we assume that the

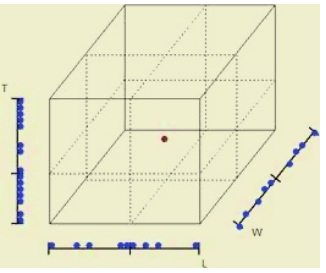


Fig. 2. A cube of tests

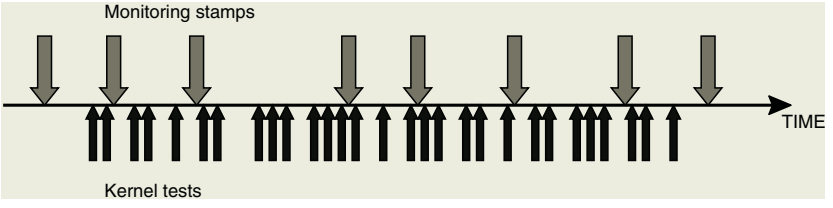


Fig. 3. Samples of the execution of the kernels in a monitorized environment

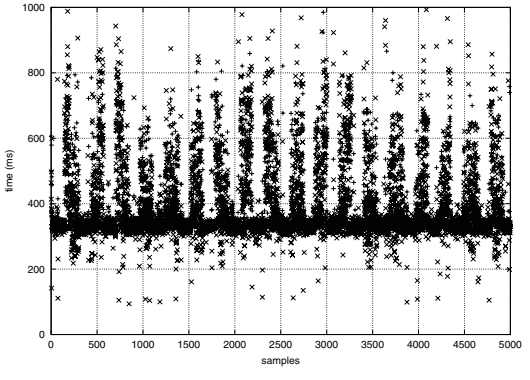


Fig. 4. 5000 samples of the execution of the ping-pong kernel

status of the system is not stable, and it can not produce a consistent cube of tests. In fact, some threshold to guaranty this homogeneity must be established to define the cubes. Figure 4 shows, as an example, the measured execution times for a ping-pong kernel executed 5000 times. The message is 32KB long. Note that most of the measures (in this case 3150) are in a thin interval of execution times, the others are not considered because they are influenced by the monitoring process, or they are considered as spurious measurements.

The monitoring process is also affected by the execution of the kernels. As an example, figures 5 and 6 show the values of, respectively, the latency and

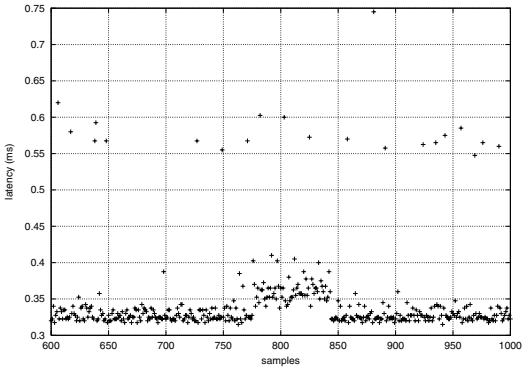


Fig. 5. Samples of the monitorized latency when the ping-pong kernel is executed

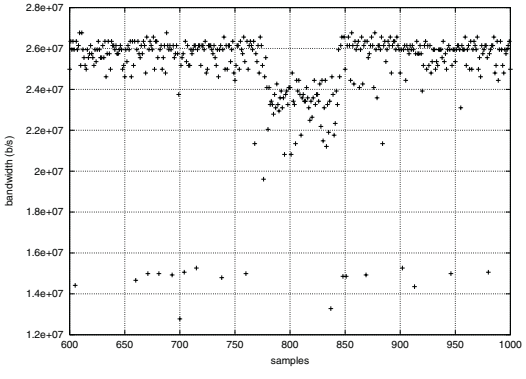


Fig. 6. Samples of the monitorized bandwidth when the ping-pong kernel is executed

bandwidth obtained from JIMS when the experiment is being performed. Note that these values change when the kernel is, in fact, executed (this corresponds to the band in the middle of the figure, around sample number 800 in the figure). Therefore the values obtained just before and after the execution of the kernel are considered to define the cube of tests. Note that some spurious values were obtained, they have to be discarded.

2.3 The Kernels

The kernels that are currently considered in the tool are the following:

– Paraiso is a MPI library of iterative methods for solving large sparse matrix systems. This library includes the implementation of analytical models to characterize a number of communications routines that can also be considered as kernels. These models for the communication routines are not only needed for implementing the models of the kernels, but also for other users who are not interested in these specific kernels.

- From the air pollution application of the CrossGrid project, we focused on the routine that consumes most of the runtime of this application. It is called vertlq. It was analyzed, and an analytical model for characterizing its performance was developed. This routine basically consists of an intensive computational part that is executed in parallel, i.e. locally, and just one reduction operation that involves communications. Both parts are uncoupled, so the model adds both contributions.
- From the flooding application of the CrossGrid project, we developed some models for the kernels of this task. For this application the standard PETSc library for solving large sparse systems is the main kernel.
- Concerning the HEP application of the CrossGrid project, this application was coded using the mater-slave paradigm. Its main kernel is a learning process of a neural network that includes two parts: a parallel part that is computational intensive, and some communications: a gather and a scatter from the master to the nodes and some point to point communications.

3 The Graphical User Interface

An interactive graphical user interface (GUI) was developed in this task. It shows three types of information:

- Information based on the analytical models, like predicted execution times, or load balance based on the predicted execution times on each node.
- Information about features of the kernel, like the number and size of some collective communication.
- Information about the status of the grid, like latency between a certain pair of nodes.

Therefore, this tool can include detailed information about specific parameters of the code, as well as the predicted information about the execution of the code. The user can interact on this information, changing some parameters, like, for example, the latency between a pair of nodes. In this way, the user can analyze their effect over the overall performance. The interface provides interactivity in the analysis of the behaviour of the code under different conditions (number of processors, distributions, input data, network parameters, ...).

This GUI is devoted to be used by applications developers and users that are interested in analyzing the performance of the selected kernels under different scenarios. The results could be used to modify the parameters of the parallel execution of the application, like the number of nodes, the size of the problem, the distribution of data, etc. To enable a user to quickly find his way in the multidimensional design space, the GUI needs to be used interactively. In long term, it can be also used by resource brokers and schedulers to select the best platform according to the predicted figures offered by the tool.

Some of the main features of this GUI are:

- It is an application dependent tool. It is specifically applied to selected kernels for which analytical models are available. Anyway, note that it can be also used by others than the CrossGrid applications developers, for example to study the performance of communication routines themselves.
- It is an interactive tool in the sense that the user can easily change the parameters that characterize the system or the code them selves, and then analyze the influence in the overall performance of these changes.
- It uses analytical models, and therefore the predictions are obtained very fast. This feature is very important for the interactivity.
- It is specifically developed for heterogeneous systems.

In summary, the GUI of the PPC tool is devoted to be used by applications developers and users that are interested in analyzing the performance of the selected kernels under different scenarios. The results could be used to modify the parameters of the parallel execution of the application, like the number of nodes, the size of the problem, the distribution of data, etc. To enable a user to quickly find his way in the multidimensional design space, PPC needs to be used interactively. In long term, it can be also used by resource brokers and schedulers to select the best platform according to the predicted figures offered by the tool.

4 Conclusions

This work present a methodology to characterize the execution of selected kernels on a Grid environment. It is based on exhaustive executions of the kernels under different states of the Grid. As a Grid can change its behavior often, we only consider measures that are homogeneous in terms of monitorized information. These pieces of homogeneous results define "cube of test" that are used to correlate the dependency with network and node based information provided by some monitoring system. This methodology can be applied to kernels that are massively computational or that are dominated by communications. A GUI was also presented in this paper.

Acknowledgement

This work was supported in part by the European Union through the IST-2001-32243 project "CrossGrid".

References

1. M. Simmons and R. Koskela. Performance instrumentation and visualization. ACM Press. 1990.
2. M. Ajmone, G. balbo and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. ACM trans. Computer Systems. Vol. 2, pp. 93-122. 1984.

3. N. Gotz, U. Herzog and M. rettelbach. Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras. Proc. SIGMETICS93.
4. V.S. Adve. Analyzing the behavior and performance of parallel programs. PhD thesis. Techn. Report 1201. Ini. Of Wisconsin. 1993.
5. V. Mak and S. Lundstrom. Predicting performance of parallel computations. IEEE trans. Parallel and distributed systems. Vols 1. pp. 253-270. 1990.
6. L. Valiant. A bridging model for parallel computations. Comm. ACM Vol 33. pp. 103-111. 1990.
7. D. Culler et.al. LogP: towards a realistic model of parallel computations. Proc. 4th ACM SIGPLAN symp. Pp. 1-12. 1993.
8. T. Fahringer. Estimating and optimizing performance for parallel programs. Computer, pp. 47-56. Nov. 1995.
9. C. Mendes and D. reed. Integrated compilation and scalability analysis for parallel systems. Proc. Int. Conf. Parallel Architectures and Compiler Technology. Pp. 385-392. 1998.
10. D.J. Kerbyson, E. Papaefstatuiou, J.S. Harper, S.C. Perry and G.R. Nudd. Is Predictive Tracing too late for HPC Users? In "High Performance Computing" Proc. HPCI'98 Conference 1998 R.J. Allan, M.F. Guest, D.S. Henty, D. Nicole and A.D. Simpson (eds.) pp 57-67. 1999. Plenum/Kluwer Publishing.
11. T. Hey, A. Dunlop and E. Hernandez. Realistic Parallel Performance Estimation Parallel Computing 23. 1997. pp. 5.21.
12. DIMEMAS. http://www.pallas.de/pages/dimemas.htm
13. K. Kubota, K. Itakura, M. Sato and T. Boku. Practical Simulation of large-scale Parallel Programs and its Performance Analysis of the NAS Parallel Benchmarks Lecture Notes Comp. Sci. 1470 1998. pp. 244-54.
14. Carnival. http://www.cs.rochester.edu/u/leblanc/prediction.html
15. J.P. Kitajima, C. Tron and B. Plateau. ALPES: a Tool for Performance Evaluation of Parallel Programs in "Environnments and Tools for Parallel Scientific Computing" J.J. Dongarra and B. Tourancheau (eds.). North-Holland. 1993. pp 213-28.
16. Fahringer Estimating and optimising performance from parallel programs. special issue IEEE Computer 28. 1995. pp. 47-56.
17. Perfore. http://ParaMount.www.ecn.purdue.edu
18. Bricks. http://www.is.ocha.ac.jp/ takefusa/bricks/
19. C.-C. Hui, M. Hamdi and I. Ahmad. SPEED: A Parallel Platform for Solving and Predicting the Performance of PDEs on Distributed Systems Concurrency: Practice and Experience 9. 1996. pp. 537-568.
20. M. Miller, C.D. Hansen and C.R. Johnson. Simulated Steering with SCIRun in a Distributed Environment in "Applied Parallel Computing" Proc. 4th International Workshop PARA'98. LNCS 1541. Springer. 1998. pp366-376.
21. PARAISO. http://www.ac.usc.es/ paraiso
22. AlpStone. www.ifi.unibas.ch/generate.doc/English/Research/ParProg/alpstone/doc.html
23. DAMIEN. http://www.hlrs.de/organization/pds/projects/damien/
24. CrossGrid project. http://www.eu-crossgrid.org/
25. PAPI. http://icl.cs.utk.edu/papi/
26. JIMS. http://wp3.crossgrid.org/

Parallel Checkpointing on a Grid-Enabled Java Platform

Yudith Cardinale and Emilio Hernández

Universidad Simón Bolívar, Departamento de Computación,
Apartado 89000, Caracas 1080-A, Venezuela
{yudith, emilio}@ldc.usb.ve

Abstract. This article describes the implementation of checkpointing and recovery services in a Java-based distributed platform. Our case study is SUMA, a distributed execution platform implemented on top of Grid services. SUMA has been designed for execution of Java bytecode, with additional support for parallel processing. SUMA middleware is built on top of commodity software and communication technologies, including Java, Corba, and Globus services. The implementation of SUMA that runs on top of Globus services is called SUMA/G.

1 Introduction

Parallel checkpointing algorithms are important in environments where long-term parallel processes are executed, for instance in grid computing platforms [7]. These algorithms can be classified into two main groups, according to the approach used to coordinate the capture of local checkpoints [6], i.e. the state of single nodes. In the coordinated approach, it is necessary to synchronize all processes in order for them to take relevant local checkpoints. This is done by sending control messages explicitly. A consistent global checkpoint, which is a global checkpoint that avoids orphan and in-transit messages, is always maintained in the system, but the synchronizing checkpointing activity results in performance degradation. In the uncoordinated approach, the processes take their local checkpoints more or less independently. As a consequence, some of the checkpoints may not belong to any consistent global checkpoint. In order to reduce the number of useless checkpoints, the processes must exchange information about its checkpointing activity. This allows the processes to take communication induced checkpoints (forced checkpoints, to avoid orphan messages) besides asynchronous checkpoints and to log possible in-transit messages. This information is mainly piggy-backed on the messages sent between processes. The algorithms based on communication-induced protocols are called quasi-synchronous [10].

We address the development of a distributed platform that transparently executes Java bytecode on remote machines, called SUMA (Scientific Ubiquitous Metacomputing Architecture) [9] (http://suma.ldc.usb.ve). The goal is to extend the Java Virtual Machine model to provide seamless access to distributed high performance resources. One of our design goals was that SUMA should provide

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 741–750, 2005.

checkpointing/recovery facilities. SUMA middleware was originally built on top of commodity software and communication technologies, including Java and Corba. A recent reimplementation called SUMA/G is also based on grid services [7], more specifically on Globus services [13], which is the most important grid model defined so far. By reimplementing SUMA components on top of Grid services we can take advantage of the potential of existing Grid platforms, and also we can keep the SUMA execution model unchanged, independent from the evolution of the Grid services. Additional functionalities are inherited from Globus, such as the I/O services. SUMA itself is implemented on top of Corba, but users of SUMA are not aware of it, they only have to write Java classes and, if they want to write a parallel program, they can use mpiJava [1], which is supported by SUMA.

Our goal is that SUMA/G supports recovery after fail-stop failures. We have implemented a distributed checkpointing facility for SUMA/G, using an uncoordinated approach based on a communication-induced protocol. The implementation of a distributed checkpointing facility on a Java distributed platform of this kind requires some special considerations related to portability, transparency, intrusiveness and persistence. In order to take into account these considerations, we propose, in this paper, a design of checkpointing and recovery services based on commodity technologies: Java, MPI and Globus.

2 Overview of SUMA/G

SUMA/G middleware is object oriented and built on top of commodity technologies. All SUMA/G components were implemented on the Java Virtual Machine, communicating via Corba [3], so it is possible to eventually implement SUMA/G components in different languages or operating systems. SUMA/G access the Globus general services through Java CoG Kit [15], in order to extend the functionality of some components (e.g. the Scheduler and the User Control). For special services (e.g. parallel execution, profiling, and checkpointing), the Execution Agents (these components actually execute the applications on the high performance platforms) could require special packages as mpiJava, Hprof and MPE, among others.

As an execution platform, SUMA/G offers the Java execution model. SUMA/G users are not aware of the presence of a distributed platform and they can invoke the execution of a class just by typing "sumag class", in the same fashion they execute that class locally (by typing "java class"). Classes and local files are sent on demand from the local machine to the actual execution platform chosen by the SUMA/G core.

2.1 Implementation of SUMA/G on Globus

Globus is distributed as a toolkit that provides basic services and libraries for resource monitoring and management, plus security, file management, job submission, and communication. There has been increasing interest in supporting Java within the grid [15, 14]. Its GT3 version, is the first full-scale implementa-

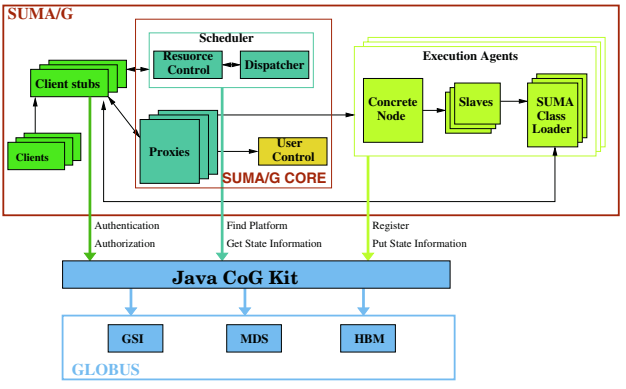


Fig. 1. SUMA/G Architecture

tion of new Open Grid Services Architecture (OGSA). However, Globus toolkits continue evolving and probably SUMA/G will need future reimplementations.

Current implementation of SUMA on top of Globus services is mainly based on the Java CoG Kit. The Java CoG Kit defines and implements a set of general components that map grid functionality into a commodity environment. The Java CoG Kit not only enables access to the Globus services, but also provides the benefit of using the Java framework as the programming model. According to the Java CoG Kit components categorization, SUMA/G mainly uses the Low-Level Grid Interface Components, that provide mappings to commonly used Grid services. The main Globus services, from the point of view of SUMA/G are (see Figure 1):

- The Monitoring and Directory System (MDS): it enables Grid application developers and users to register their services with the MDS. The Globus MDS service is used by SUMA/G for component registration. In this way the SUMA/G scheduler, for example, queries the MDS to find a suitable platform when it receives a request for a particular job execution. With this service, resources that are allocated by Globus can then be added to the SUMA/G resources, and SUMA/G will be able to deal in most cases with organizational boundaries. SUMA/G has its own mechanism for submission and allocation of resources, but we expect to achieve a greater integration of SUMA/G with GRAM (Globus Resource Allocation Manager).
- The Globus Heartbeat Monitor (HBM): it provides a simple, uniform and reliable mechanism for obtaining real-time information about the grid structure and status. This includes monitoring the state of machine and processes in the grid. This mechanism allows SUMA/G Execution Agents to post information about their state, and SUMA/G core could receive that information in order to make scheduling decisions.
- The Grid Security Infrastructure (GSI): it defines a common credential format based on X.509 identify certificates and a common protocol based on

transport layer security (TLS, SSL). GSI policy allows a user to authenticate just once per computation, at which time a credential is generated that allows processes created on behalf of the user to acquire resources, and so on, without additional user intervention. The initial security component in SUMA is based on a simple centralized maintenance based on user accounts and user groups known in a typical UNIX system. This security component is not strong enough to support the increased security requirements in a grid environment. The GSI is used to augment the user authentication and authorization in SUMA/G, as well as a mechanism for including all SUMA/G components in the grid security space.

3 Checkpointing and Recovering Services in SUMA/G

SUMA/G is intended to provide services for checkpointing and recovery, apart from basic services related to on-line and off-line remote execution of Java applications. These services are mainly implemented within the Execution Agents. In this way, SUMA/G provides different Execution Agents with particular services. In order to provide checkpointing and recovery services for single-node and parallel Java applications, an Execution Agent with checkpointing support should fulfill the following demands:

- Capacity for permanent storage. The checkpoints can be stored either in local storage or in a remote checkpoint server. The first option could be implemented through the access to a shared file system, such as NFS.
- Execute the extended JVM for providing local checkpointing/recovery services at threads level.
- Execute the modified mpiJava that allows access to the messages exchanged through MPI in order to have Global Consistent Checkpoints.

Figure 2 shows the new SUMA/G architecture after checkpointing/recovery services were incorporated. There are two important components: `SUMACkpMonitor` and `SUMARecover`. In the current implementation, the Execution Agent starts the application, the `SUMAClassloader` and the thread `SUMACkpMonitor` in the extended JVM. `SUMACkpMonitor` periodically takes the checkpoints, in an asynchronous way. In case of parallel applications, this thread additionally is in charge to take forced checkpoints when it is necessary. On the other hand, the `SUMARecover` is called after a fault occurs and is in charge of restoring the application execution from its last checkpoint.

The level of transparency that a checkpointing system can offer to the users can be considered from two points of view. First, it is important to decide whether modifications of the application source code are necessary, for instance, by adding threads or data structures that are not related to the application. To reach this goal, the process to capture the execution state is executed by `SUMACkpMonitor`, without involving the application itself. Second, it is important to consider whether the user should have control on the restarting and

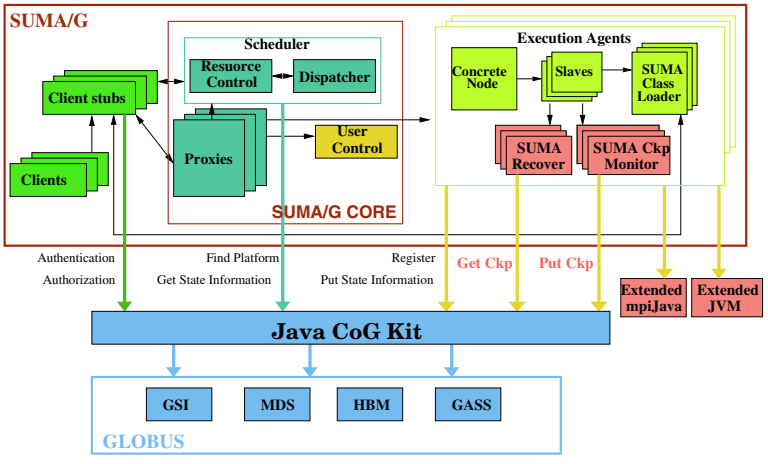


Fig. 2. SUMA/G Architecture with Globus support for checkpointing/recovery services

migration actions. SUMA/G´s goal is to support recovery after fail-stop failures in a transparent way. In this sense, after a crash occurs, SUMA/G launches the recovery process without user involvement.

In order to take local checkpoints, we have used the implementation of Java Thread Mobility and Persistence inside the JVM proposed in [2], which consists on extending the JVM in such a way that the computation state is accessible from Java threads. The reasons that led us to choose this approach and the implementation details in SUMA are described in [4], where it is shown that this approach does not introduce significant performance overhead.

Our proposed distributed checkpointing protocol is a combination of the protocols described in [11] and [8] which provide fault-tolerance in asynchronous systems. We have extended the mpiJava functionality to implement the combined checkpointing protocol. The details of the algorithm and results about the implementation overhead are described elsewhere [5].

3.1 Checkpointing Support in Globus

Globus does not provide specific services for checkpointing/recovery, but it provides basic services, such as the Globus Heartbeat Monitor, that could help us to implement checkpointing and recovery services. Specific tools implemented on the grid, such as Condor, which has centralized checkpointing facilities [12], could also be helpful in order to implement a distributed checkpointing and recovery facility. Other Globus components can be used to implement checkpointing/recovery services, such as the I/O services.

After a crash, SUMA/G has to determine an Execution Agent on which the application can be re-initialized. To do so, the SUMA/G core asks the Globus MDS for another computing platform suitable for that application (i.e. with the appropriate extended JVM and extended mpiJava installed). One the SU-

MA/G core has identified the new platform, it has to transfer the last checkpoint to the selected machine. Then the application can be re-started on that machine.

The heterogeneity of mechanism and policy encountered in grid environments means that we can not assume that the computation platforms (where the SUMA/G Execution Agents run) share a file system, user id space, or common security mechanisms. Globus provides a uniform naming scheme and the same access mechanism to access files, regardless of location with Global Access to Secondary Storage (GASS) services. In order to keep the checkpoints, SUMA/G can either store the checkpoints on top of virtual file platforms that handle Logical File Names (LFN) or in a particular Checkpoint Server directly accessible through GASS services. For the sake of simplicity, we are going to assume that there is a Checkpoint Server for checkpoint storage and management. The Java CoG Kit provides an essential subset of GASS services to support the copying of files between computers on which the Grid Services are installed. The method `get(String from, String to)` copies a remote file to a local file, and the method `put(String from, String to)` copies a local file to a remote location. The `SUMACkpMonitor` posts the checkpoint files periodically to the Checkpoint Server with `URLlocalckp.put(localhost,CkpServer)`, and the `SUMARecovery` gets the checkpoints from the Checkpoint Server with `URLlocalckp=get(CkpServer,localhost)`.

The GASS open and close calls act like their standard UNIX I/O counterparts, except that a URL rather than a file name is used to specify the location of the file data. A URL used in a GASS open call specifies a remote file name (a checkpoint, in case of SUMA/G), the physical location of the data resource on which the file is located (e.g. the SUMA/G Checkpoint Server), and the protocol required to access the resource (e.g. GASS, ftp, or http protocol).

4 Recovery Algorithm

If a failure occurs in the platform while an application is running, an exception is caught in the SUMA/G core. In this case, SUMA/G launches a recovery algorithm. Figure 3 shows the execution of the recovery algorithm in three cases: (a) when checkpoints are stored on local disk, in this case the application could be restarted at the same platform, (b) when checkpoints are stored on a shared File System, in this case the application could be restarted at any platform sharing the file system, and (c) when the checkpoints are stored in a Checkpoint Server based on Globus facilities, so the recovery can take place in any remote platform reachable by SUMA/G.

In case of multiple-nodes applications, the last consistent global checkpoint is not necessarily formed by the last local checkpoint of each process. In this sense, the method `getlastckpt` of the Execution Agent and the method `Getckpt` of the Checkpoint Server need to associate an algorithm to select the last consistent global checkpoint. Hence, the previous algorithms must be complemented with the following sentences:

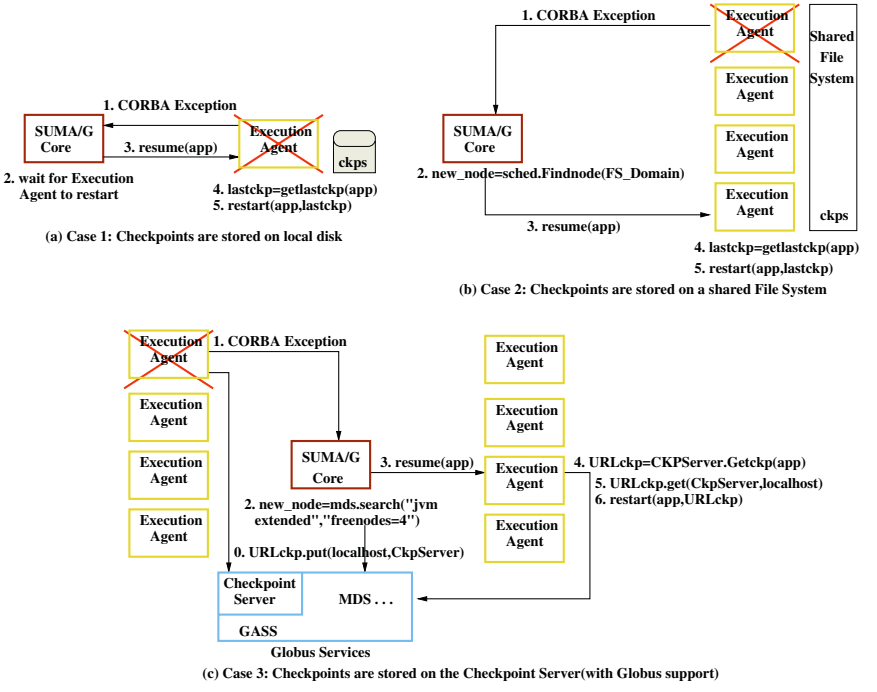


Fig. 3. SUMA/G Recovery Algorithms

- Non-faulty processes are required to take a local checkpoint (this allows as much correct computation as possible to be saved)
- **In `getlastckpt` and `GetCkp` methods**: The most recent consistent global checkpoint is determined. This determination is done in the following way:
  - obtain the last timestamp a that is common in almost one local checkpoint of each process.
  - built a consistent global checkpoint Ca=$C_{1,x1}$,$C_{2,x2}$, . . ., $C_{n,xn}$ defined as:
    ($C_{i,xi}$ is the local checkpoint of P_i)
    $\forall$k, $C_{k,xk}$ is the last checkpoint of P_k such that $C_{k,xk}$.t <= a
- **In Execution Agent `Restart` method**: The state of each channel c_{ij} (messages in transit with respect to the ordered pair ($C_{i,xi}$,($C_{j,xj}$)) is extracted from the stable storage of P_i and re-sent to P_j.

5 Experimental Results

We implemented the previous algorithms in SUMA/G and conducted experiments to evaluate the checkpointing intrusiveness and the overhead of the recovery process. The platform used in the experiments was a set of clusters of 143 MHz SUN Ultra 1 workstations running Solaris 7, with 64 MB Ram and a 32 Kbytes cache memory, and a 10Mbps switched Ethernet LAN.

In order to evaluate checkpointing intrusiveness, we measured the total execution time with (T_{ckp}) and without (T_{nockp}) invoking the checkpointing service. The checkpointing overhead percent is given by:

$$O\%_{ckp} = \frac{(T_{ckp} - T_{nockp})}{T_{nockp}} * 100$$

The time spent by the recovery processes is measured by the SUMA/G core, between the point in which the failure exception is received and the point in which the "resume" service returns. The failure is simulated by killing the processes. In all cases the application was restarted in a different execution platform.

We tested the checkpointing and recovery services on two kind of applications: single-node and multiple-nodes programs. On single-node small applications the overhead introduced was very low, typically less than 5%. Table 1 shows the total execution time with and without checkpointing service for a numeric application called "Acoustic", which solves an acoustic wave propagation model on an homogeneous, two dimensional medium. The problem size depends on the plain dimensions and the total time, so it's kept constant during the execution. The checkpoints were taken every four minutes. For the "Acoustic" application, the overhead is less than 0.5%.

Table 1. Total Execution Times for the Acoustic program

$T_{nockp}(1)$	$T_{ckp}(2)$	# of ckps	ckps size	(2) - (1) $O\%_{ckp}$
103 min 29 sec	103 min 51 sec	25	2 KB	(22 sec) 0.35%

We also executed several multiple-nodes programs. The first program is a small synthetic program that calculates π, called "Pi_Number". All processes execute the same amount of work, regardless of the number of processes used, and keep an array of data obtained during the execution, so the checkpoint size increases during the execution. The second one is the "Acoustic" parallel version (called "Acoustic_Par") and its checkpoint size is constant during the execution.

Table 2 shows the measurements with and without the checkpointing service for the parallel programs. Each row represents a single execution. For all cases, the checkpoints were taken every 2 minutes approximately. The overhead ($O\%_{ckp}$) exhibited when the checkpoint service is used increases with the number of processors. This is mainly due to the checkpoint calls and the distributed checkpointing protocol. On the other hand, the "Acoustic_Par" application reduces the execution time as the number of processors is increased.

We can note that the overhead for parallel applications is greater than the overhead for sequential programs for several reasons: 1) there are more update activities due to the distributed checkpointing algorithm, 2) on each checkpoint, it is necessary to save additionally, the messages in transit, 3) all checkpoints are saved in NFS (all processes contact the same file server), and 4) the applications take a short time, so they are sensitives to the checkpointing intrusiveness.

Table 2. Total execution time for the multiple-nodes programs

App name	# of proc	$T_{nockp}(1)$	$T_{ckp}(2)$	Checkpoints			(2)-(1) $O\%_{ckp}$
				#	min size (KB)	max size (KB)	
Pi_Number	2	6.74 m	7.1 m	3	1	131	0.36 m - 5.1%
	4	7.31 m	7.77 m	3	1	139	0.48 m - 5.92%
	6	7.68 m	8.18 m	4	1	142	0.5 m - 6.11%
Acoustic Parallel	2	5.45 m	5.51 m	2	1	1	0.06 m - 1.1%
	4	3.56 m	3.65 m	2	1	1	0.09 m - 2.47%
	8	2.64 m	2.71 m	2	1	1	0.07 m - 2.58%

Table 3 shows checkpointing and recovery overhead for distributed applications. In each case the application was recovered from a checkpoint with the same size but a different number of processors. Note that the recovery extra time % increases as the number of processors is increased. It clearly depends on the size of the checkpoints taken and the number of processors.

Table 3. Checkpointing and recovery overhead for the multiple-nodes programs

App name	# of proc	Ckp size (KB)	$\overline{M_{intransit}}$	Ckp time (sec)	Recov time (sec)	% of extra time
Pi_Number	2	65	220B	1,71	5,51	68.97
	4		255B		6,45	73.49
	6		410B		7,27	76.48
Acoustic Parallel	2	1	3KB	51 msec	4,08	98.75
	4				5,65	99.21
	8				6,38	99.30

6 Conclusions and Future Work

This work addressed important aspects related to the implementation of a checkpointing facility on a distributed execution platform intended to be implemented on top of grid services, such as SUMA/G. This approach is almost transparent at the user level, because the requirements for using this facility can be easily hidden by using pre-processor and SUMA/G stubs.

We present an implementation of a distributed algorithm based on a communication-induced checkpointing protocol (quasi-synchronous algorithm). The algorithm was implemented for execution of distributed Java bytecode using *mpiJava*, a wrapper for MPI, and tested on a grid-enabled Java execution platform. The checkpointing/recovery facility is automatic if the parallel Java program uses our extended *mpiJava* library and SUMA/G is used as the execution platform, rather than submitting the *mpiJava* job directly on Globus execution servers.

In order to adequately implement the proposed checkpointing facility, some basic grid services may be useful for reducing the checkpoint capture overhead. A Heartbeat Monitor was not strictly necessary for this implementation. Instead, it seems more important the implementation of I/O support specifically oriented to achieve a safe management of checkpoints. This mechanism could be implemented as a Checkpoint Server accessible through GASS or, alternatively, the checkpoints may be stored as virtual files in the grid. With the experiment

results, we concluded that, for the distributed checkpointing scheme that we use, checkpoint manipulation dominates the overhead when using such facility, rather than the synchronization operations needed among the processes.

References

1. M. Baker, B. Carpenter, S. Hoon Ko, and X. Li. mpiJava: A Java interface to MPI. In *First UK Workshop on Java for High Performance Network Computing, Europar 98*, 1998.
2. S. Bouchenak. Making Java applications mobile or persistent. In *Proceedings of 6th USENIX Conference on Object-Oriented Technologies and Systems*, 2001.
3. Y. Cardinale, M. Curiel, C. Figueira, P. García, and E. Hernández. Implementation of a corba-based metacomputing system. *Lecture Notes in Computer Science*, 2110(Java in High Performance Computing), June 2001.
4. Y. Cardinale and E. Hernández. Checkpointing facility in a metasystem. In *Lecture Notes in Computer Science. 7th International Euro-Par Conference*, volume 2150, Manchester, UK, August 2001. Springer Verlag.
5. Y. Cardinale and E. Hernández. Parallel checkpointing facility in a metasystem. In *Proceedings of The Parallel Computing Conference*, Naples, Italy, 2001.
6. E. N. Elnozahy, L. Alvisi, Y-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(30), 2002.
7. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3), 2001.
8. J. M. Helary, A. Mostefaoui, R Netzer, and M. Raynal. Communication-based prevention of useless checkpoints in distributed computations. Technical Report Publication interne n 1105, Institut de Recherche en Informatique et Systemes Aleatoires, May 1997.
9. E. Hernández, Y. Cardinale, C. Figueira, and A. Teruel. SUMA: A Scientific Metacomputer. In *Parallel Computing: Fundamentals and Applications. Proceedings of The International Conference*. Imperial College Press, 2000.
10. D. Manivannan and Mukesh Singhal. Quasi-Synchronous Checkpointing: Models, Characterization, and Classification. *IEEE Transactions on Parallel and Distributed Systems*, 10(7), July 1999.
11. A. Mostefaoui and M. Raynal. Efficient message logging for uncoordinated checkpointing protocols. Technical Report Publication interne n 1018, Institut de Recherche en Informatique et Systemes Aleatoires, June 1996.
12. G. Stellner. Cocheck: Checkpointing and process migration for MPI. In *10th International Parallel Processing Symposium*, 1996.
13. The Globus Alliance. The Globus Toolkit. http://www.globus.org/.
14. The Globus Alliance. The Globus Toolkit. http://www.globus.org/ogsa.
15. G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM Java Grande 2000 Conference*, San Francisco, CA, JUNE 2000.

Fault Tolerance in the R-GMA Information and Monitoring System

Rob Byrom[4], Brian Coghlan[6], Andy Cooke[1], Roney Cordenonsi[3], Linda Cornwall[4], Martin Craig[4], Abdeslem Djaoui[4], Alastair Duncan[4], Steve Fisher[4], Alasdair Gray[1], Steve Hicks[4], Stuart Kenny[6], Jason Leake[7], Oliver Lyttleton[6], James Magowan[2], Robin Middleton[4], Werner Nutt[1], David O'Callaghan[6], Norbert Podhorszki[5], Paul Taylor[2], John Walk[4], and Antony Wilson[4]

[1] Heriot-Watt University, Edinburgh
[2] IBM-UK
[3] Queen Mary, University of London
[4] Rutherford Appleton Laboratory,
[5] SZTAKI, Hungary
[6] Trinity College Dublin
[7] Objective Engineering Ltd.

Abstract. R-GMA (Relational Grid Monitoring Architecture) [1] is a grid monitoring and information system that provides a global view of data distributed across a grid system. R-GMA creates the impression of a single centralised repository of information, but in reality the information can be stored at many different locations on the grid. The Registry and Schema are key components of R-GMA. The Registry matches queries for information to data sources that provide the appropriate information. The Schema defines the tables that can be queried. Without the combined availability of these components, R-GMA ceases to operate as a useful service. This paper presents an overview of R-GMA and describes the Registry replication design and implementation. A replication algorithm for the Schema has also been designed.

1 Introduction

R-GMA is a monitoring and information application for grid environments. It is an implementation of the Grid Monitoring Architecture [2] proposed by the Global Grid Forum [3]. R-GMA was developed within the European DataGrid [4] project. . It is currently being re-engineered as web services, within the EGEE (Enabling Grids for E-Science in Europe) project.

It has been used as the information and monitoring service within the DataGrid project, where it provided information on Computing and Storage Elements which was then used by the Resource Broker to decide where to submit jobs. Network monitoring data were also published using R-GMA [5]. A description of the use of R-GMA within DataGrid, along with detailed performance measurements (including indications of the performance gain with replicated registries) is about to be published [6].

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 751 – 760, 2005.

The LHC Computing (LCG) project [7] aims to meet its computing needs by deploying a grid composed of computing resources distributed across many different countries. R-GMA is being used to implement an accounting service on LCG.

Santa-G (Grid-enabled System Area Networks Trace Analysis) was used to develop a network traffic monitoring application within the CrossGrid [8] project. This NetTracer application publishes information using R-GMA that can be used for the validation and calibration of intrusive monitoring systems, and also for analysing the performance of a site in a network.

2 Grid Monitoring Architecture

The Grid Monitoring Architecture models the information infrastructure of the grid as a set of Producers that provide information, and a set of Consumers that request information. A Registry contains details of Producers and Consumers. Producers contact the Registry to announce the structure of the data they provide. Consumers contact the Registry to find Producers that publish information they require. The Registry returns to the Consumer a list of Producers that provide the desired information, and the Consumer then contacts the appropriate Producer or Producers to obtain this information.

3 Virtual Database

R-GMA presents a global view of the information produced by the components of a grid system and by applications running on the grid. It presents this information as a virtual database, introducing the relational data model to the Grid Monitoring Architecture. Information can be inserted into and retrieved from the R-GMA virtual database using a subset of the statements specified in the SQL92 Entry Level standard. To a user who queries this virtual database, it seems as though they are querying a single, centralised repository of information. In actual fact, the information obtained from this virtual database can be stored at a number of different locations.

Each table has a *key* column (or group of columns). In addition, each tuple inserted by a Producer has a timestamp added to it by the Producer. The combination of this timestamp and the key columns is similar to a primary key for the table. This enables R-GMA to provide time-sequenced data to monitoring applications where users may wish to request data published within a particular time interval. Of course for this to be reliable, all systems in an R-GMA installation must synchronize their system clocks using a tool suc as NTP.

4 Producer Service

The Producer service is used to publish data to the virtual database. The R-GMA schema contains the definitions of all tables in the virtual database, and Producers declare their intention to publish to a table by registering themselves with the Registry. The Producer service obtains the table structure from the Schema and

creates a table with identical structure in its own storage. Data in the Producer's storage has a retention period associated with it. When data has been in the storage for longer than this retention period, it is deleted. The length of this retention period is dependent upon how long users require access to tuples after they have been published. There are three kinds of Producers. The difference between them is where the data they publish is coming from.

- *Primary Producer:* The user's code inserts tuples directly into the Producer's storage.
- *Secondary Producer:* The Secondary Producer republishes tuples that have already been published by another Producer. It queries Primary Producers and other Secondary Producers and inserts the tuples returned from these queries into its own storage. This can be useful for creating a service that archives the data from multiple Producers, or for answering queries that require joins by combining tables from multiple Producers.
- *OnDemand Producer:* It may not be practical to transfer a large volume of data to a Producer with SQL INSERT statements. The OnDemand Producer allows a user to request such information when it is required. Only then is the information encoded in tabular form and sent to the user. The OnDemand producer has no tuple storage facility. It sends all queries it receives to additional user code which sends tuples back to the consumer.

5 Consumer Service

Consumers allow users to execute SQL queries on the R-GMA virtual database. The Consumer contacts the Registry to find the Producers required to answer the query. It then sends the query to these Producers and collects the tuples returned into an internal store for subsequent retrieval by the user. Consumers allow four types of query:

- *Continuous:* All tuples matching a query are to be automatically streamed to the Consumer when they are inserted into the table. All Primary and Secondary Producers support continuous queries, but OnDemand Producers do not. An example of where receiving a continuous stream of data from a producer is of use would be a real-time job monitoring application, which must update the status of jobs on the grid as they are executing.
- *Latest:* The most recent version of a tuple matching a query is returned to the user. A resource broker which decides where jobs should be executed on a grid may use a Latest producer. The resource broker needs up-to-date information about the resources on the grid. It is only interested in the most recent state of the resources, not their state over a period of time.
- *History:* All available tuples matching a query are returned. This is useful when the Producer is to be used as part of an archiving application.
- *Static:* These are specific to OnDemand Producers, and are handled like a normal database query (no timestamps or intervals associated with these queries).

6 Registry Service

The Registry enables R-GMA to match Consumers to Producers that provide the information they require. Producers advertise what tables and parts of tables in the virtual database they produce rows for. Consumers can consult the registry to find Producers that can answer their queries. The process by which the Registry finds Producer(s) that are capable of providing information requested by a Consumer is called mediation[9]. Mediation allows a Consumer to have a global view of information from sources distributed across the grid.

7 Schema Service

R-GMA uses a Schema service to define the tables that comprise the virtual database. It also defines the authorization rights for these tables. The Schema service allows operations such as adding or removing tables to/from the virtual database, or getting the name, type and attributes of all columns in a particular table.

8 Namespaces

The term "Virtual Organisation" (VO) [10] has been formulated to represent a set of resources that are shared by a number of users, and rules that specify users' access rights to resources. Individuals and institutions working on the same problems (for example, a community of researchers working in a particular area, such as High Energy Physics or Earth Observation), benefit from pooling their resources and making them available to all members of the community. It is possible that a user can be a member of more than one VO.

R-GMA allows users to issue queries to multiple VOs. A set of VOs may contain semantically different tables with identical names. Clearly a global namespace defined by just the table names is not scalable. The R-GMA solution is to form a virtual global namespace by giving each VO its own disjoint subset of the namespace. Information particular to each VO is contained in a virtual database, the name of the virtual database being that of the VO. Although it is possible to have a single registry and schema, scalability concerns strongly favour that each VO has a separate set of schema and registry services. To issue a query to a particular VO using a consumer, the table name in the SQL query must be prefaced with the VO name. For example, for the LCG VO:

```
SELECT NumberOfJobsRunning FROM LCG.Workload WHERE
Site='RAL'
```

For publishing information we could adopt the same syntactic approach. However as it is common to wish to publish the same information to multiple VOs, the set of VOs that a tuple is published to is included as a separate parameter in the Producer API function call, and is not included in the SQL INSERT statement.

9 Registry Replication

Although there is only one logical Registry per VO, replicas of the Registry are maintained. If a particular Registry fails at any time, an alternative Registry is used instead. This enables the client (i.e. the Consumer or Producer) to carry out a Registry operation in spite of any faults that occur within individual components of the replicated Registry group.

9.1 Selecting a Registry

Clients wishing to connect to a Registry do so by using a selection algorithm, which is carried out internally within the Registry API. A list of available Registry services is provided via a configuration file on the client. During runtime, a simple profile is created by contacting each Registry service identified in the configuration. The quickest response time is then used to select the Registry to handle the client request. This Registry will then be re-used for all further client interaction unless the Registry becomes unavailable - at which point a new one is selected using the same procedure.

A reliable Registry Service can be demonstrated with only a small set of Registries, typically between 2 to 4 per VO. Any overhead incurred from contacting each Registry is therefore acceptable for small groups of Registry peers. A more sophisticated profiling algorithm, perhaps based on the load average or overall reliability of a Registry, might benefit larger installations.

9.2 Registry Replication Design

Registry replication is based on a distributed model where each Registry pushes locally acquired data to its peers. New data is obtained when a Consumer or Producer carries out a registration process. When this occurs, an entry is copied into the local Registry database along with a *replica status* flag indicating the data is fresh. Conversely, the *replica status* flag is set to delete when a Consumer or Producer is closed by the client. A further state referred to as “replicated” is encoded by the *replica status* flag and represents rows that were copied in a previous replica update. An additional *origin* tag is added to each entry that identifies the Registry where the new data was added.

A dedicated thread that runs periodically on each Registry replica initiates the replication process. It checks the existence of newly arrived registrations by reading the replica status flag of entries in the Registry. An additional check is made to ensure the origin matches the current Registry so only local data is considered.

Once a list of candidate data is identified for replication, a replica message is constructed which encodes all the records within an XML format. XML was chosen due to open source API’s which provide XML parsing functionality. Although XML imposes additional padding to a replica message, the schema adopted has been carefully designed to help minimize the message size. On average, XML encoding increases the replica message by a factor of 1.5.

The Registry reads from a configuration file the URLs of all Registry and Schema Servers that are members of the same VO. When the XML message is constructed, a checksum is then computed based on the state of the Registry and the final replica

message is then sent to each Registry peer. A list of available Registries is read-in from a configuration file each time the replication cycle is scheduled. When a replica message is received, the message is passed through a filter that works out in which table each record should be stored or removed. Once each record in the replica update is processed, the checksum is re-calculated to validate the consistency of the databases between the sender and recipient. If the checksums match up, a successful response message is returned to the sender. At that point, the *replica status* flag of all the copied records on the sending registry are set to "replicated" to prevent further replication in future.

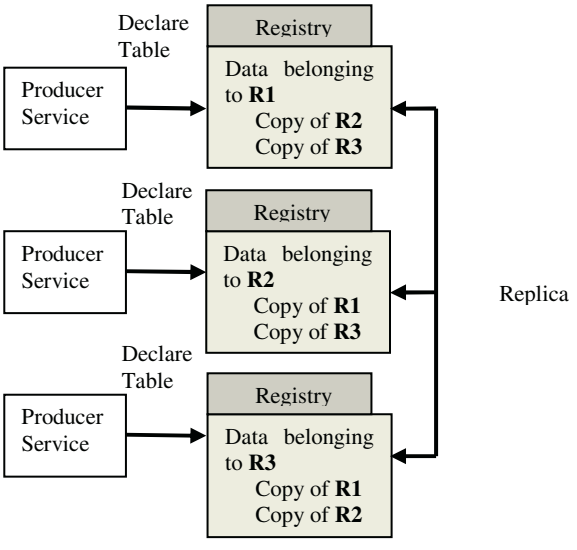


Fig. 1. Only data local to a Registry is replicated

If the checksums do not match, the replication attempt has failed, indicating that the data stored within the sender's database is inconsistent with that of its peer. In order to resolve this conflict, the original sender will copy all records that match the Registry's origin irrespective of whether the record has been replicated or not. The recipient Registry will then delete all records that match the sender's origin and then perform a clean install using the new replica message. This therefore ensures a high level of consistency is preserved, and that errors cannot go unnoticed.

A checksum failure means a Registry may have provided inaccurate mediation prior to correction. In the worst case, a Producer or Consumer may receive false information about a resource that no longer exists. In this case, the Producer and Consumer Service are robust enough to quietly dispose of the false information.

When a replica update is successfully stored (whether it is a complete or partial update) the receiving registry will attempt to mediate newly replicated Consumers with any locally stored Producers which do not match the sender's origin. If a complete update is made, then it is possible a Consumer may receive duplicate calls

for the same producer. If this occurs, the Consumer Service will simply ignore the duplicate. The mediation phase ensures the query plans used by the Consumers and Secondary Producers are therefore complete.

9.3 Batching Updates

A Registry replication update is executed as one large batch. While this approach helps to reduce the overall load on each Registry, some inconsistency is introduced between successive batch updates. However, the Registry copes with any potential inconsistencies by performing an extra mediation phase once a batch update is received. As mentioned previously, this mediation phase ensures that any Consumers that may have missed a Producer are then notified. In the worst case, a Consumer may not receive data until the next batch update. By default, this is set to an interval of 30 seconds but can be re-configured. This approach seems to work, but scalability may require the replication interval to be automatically adjusted, based on Registry usage.

10 Schema Replication Design

We have considered several possible designs for a schema replication system, mostly based around master-slave or atomic commitment protocols. All have struggled to balance the requirements of global consistency, fault tolerance and scalability. In fact we can relax the first of these. Schema replicas don't have to be identical for R-GMA to operate safely, it is only necessary that individual table definitions (column names and types) be uniquely defined. We do permit table names to be re-used (dropped and re-created), but tables are identified internally by a *table number* which is unique for all time, and we don't permit table definitions to be modified. Since mediation is done on table numbers, not names, there is no possibility of the mediator matching up producers and consumers that have different ideas about a table's definition. Of course permitting schema replicas to get temporarily out of step (which could result in consumers not being matched up with producers, and vice versa) is not ideal, but it is safe, and in practice it is very unlikely to occur because dropping and re-creating tables is a rare event, especially on tables that are still being used elsewhere in the VO. With these observations, the replication problem reduces to designing an allocation system for unique table numbers for *new* tables, along with a simple scheme to re-synchronize replicas in slower time, following any changes.

10.1 Schema Write Operations

Changes to the schema are, of course, visible across the whole VO. We can therefore assume that schema write operations will require special privileges, will be relatively rare, and will probably be initiated by hand. We'll see that these assumptions influence some of the design choices we make in the replication design. The write operations are *createTable*, *dropTable* and *alterTableAuthorization*. It turns out that it is convenient, for re-synchronization, to allocate a unique version number to the schema itself which is changed following any write operation. It can also be used as the new table number for a *createTable* operation. Thus the algorithm for all schema write operations is broadly the same:

1. Obtain a write lock on the schema database.
2. Agree a new schema version number with the other replicas, as described below.
3. Commit the schema changes and the new version number to the database, as a single transaction (so schema readers see an atomic change).
4. Release the database lock.
5. Schedule an immediate re-synchronization with the other replicas.

Read operations on the schema are allowed to continue throughout this process.

10.2 Schema Version Number Allocation

Each schema replica has a *version* property and a *nextVersion* property stored in its database. These are both integers, initialised to zero when the replica is first created. Each replica also has access to a list of URLs of all other replicas. Finally each has a *newVersion(proposedVersion)* operation called by other replicas when they want it to agree a new version number, defined as follows:

1. Obtain a write lock on the local schema database.
2. Get the current value of *nextVersion*.
3. If (*nextVersion* < *proposedVersion*)
   Set *nextVersion* to *proposedVersion* and commit it to the database.
   Else
   Set *nextVersion* to (*nextVersion* + *1*) and commit it to the database.
   End If.
4. Release the database lock.
5. Return *nextVersion* to the caller.

When a replica wants to agree a new schema version number, it first increments its own *nextVersion* number and commits it to its database, then proposes it to all other replicas by calling *newVersion()*, repeating the process until agreement is reached. Although a synchronous call to each replica is potentially slow, it is a reasonable approach for a manually initiated operation, given the relatively small number of replicas a VO is likely to require. A time-out is used to make sure the call to each replica completes or fails in a reasonable length of time, and only a majority of replicas are required to reply for the process to succeed. The exact procedure is as follows:

1. Increment the *nextVersion* property to (*nextVersion* + *1*) and commit it to the database.
2. Loop through the list of replicas, sending a *newVersion(nextVersion)* request to each of them.
3. If (fewer than (N + 2) / 2 reply, for N replicas)
   Abandon the operation.
   Else if (all replicas accept *nextVersion*)
   Use *nextVersion* as the new schema version number.
   Else if (this is the second attempt to obtain a new version number)
   Abandon the operation.
   Else
   Set *nextVersion* to the highest version number returned by any replica.
   Commit *nextVersion* to the database, and try again.
   End If.

It is essential that a replica always increments its *nextVersion* property before proposing it to the other replicas, so that concurrent instances of this algorithm don't end up with the same number. There is still a possible race condition where two simultaneous instances contend with each other for a new version number and the process never terminates, but again, since schema write operations are manually initiated, it is sufficient just to make a limited number of tries before giving up with a suitable error message.

10.3 Synchronizing Replicas

A replica synchronizes itself with another replica by sending a synchronization request containing its own *version* number to the remote replica. The remote replica returns its *version* number in reply. If the remote replica is at a higher version number, it also returns a copy of itself with a checksum, much as in the registry replication. If, it is at a lower version number, it schedules its own synchronization request. A replica sends synchronization requests to all other replicas when it first starts up (selecting the most up to date reply) and when it completes a write operation. It also periodically sends re-synchronization requests to any replicas which it believes to be out of date (based on an in-memory list), to prompt them to re-synchronize themselves.

10.4 Notifying Producers and Consumers of Schema Changes

Normally, producer and consumer instances don't need to be notified about schema changes, because they retain their own copy of table definitions for any tables on which they are currently processing queries, and these table definitions are immutable. Table authorization can change, however, and since producer and consumer services are responsible for imposing table authorization, these changes may need to be propagated immediately.

11 Conclusions

R-GMA provides a global view of information produced by applications distributed over a grid. The Registry and Schema are key components of R-GMA. If these components are unavailable, R-GMA ceases to operate as a useful information and monitoring service. Replication algorithms for both the Registry and Schema have been designed. If one of these components fails at any time, an alternative working Registry/Schema can be used instead. When replicas of the Registry and Schema are created, these components are no longer single points of failure within R-GMA. This improves the scalability and fault tolerance of R-GMA.

References

1. A. Cooke, A. Gray et al., *R-GMA: An Information Integration System for Grid Monitoring*, in Proceedings of the Tenth International Conference on Cooperative Information Systems (2003).

2. B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski and M. Swany, *A Grid Monitoring Architecture*, GGF 2001. http://www-didc.lbl.gov/ggf-perf/gma-wg/papers/gwd-gp-16-3.pdf.
3. Global Grid Forum: http://www.ggf.org/
4. DataGrid project: http://www.edg.org/
5. F. Bonnassieux, *Final Report on Network Infrastructure and Services*, deliverable for DataGrid Project, 2003.
6. A.Cooke, A.Gray et al., *The Relational Grid Monitoring Architecture: Mediating Information about the Grid*, to be published in Journal of Grid Computing
7. LHC Computing Grid Project: http://lcg.web.cern.ch/
8. CrossGrid project: http://www.crossgrid.org/
9. A. Cooke, A.J.G. Gray and W. Nutt, *Stream Integration Techniques for Grid Monitoring,* Journal on Data Semantics, 2, 2004. LNCS 3360
10. I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid: Enabling ScalableVirtual Organizations*, International Journal of High Performance Computing Applications. 15(3), 2001

Deployment of Grid Gateways Using Virtual Machines

Stephen Childs, Brian Coghlan, David O'Callaghan, Geoff Quigley and John Walsh

Department of Computer Science,
Trinity College Dublin, Ireland
{childss, coghlan, ocalladw, gquigle, walshj1}@cs.tcd.ie

Abstract. Grid-Ireland, the national computational grid for Ireland, has a centrally managed core infrastructure: the installation and configuration of grid gateways at constituent sites are controlled from an Operations Centre based at Trinity College Dublin. We have developed tools to automate and simplify the deployment of Grid middleware to these sites. Virtual machine (VM) technology has performed an important role, allowing us to maximise the utilisation of server hardware and to simplify installation and management procedures. By running multiple OS instances, each on a VM, a full LCG-compatible Grid gateway can be hosted on a single computer. This has significantly reduced the hardware, installation and management investment needed to deploy a new site. In this paper, we summarise an evaluation of competing VM technologies and relate our experience with virtual machines to date. We also describe a single-computer Grid gateway based on the Xen VM system which we plan to deploy to eleven new sites in early 2005.

1 Introduction

1.1 Context

The Grid-Ireland project provides grid services above the Irish research network, allowing researchers to share computing and storage resources using a common interface. It also provides for extra-national collaborations by linking Irish sites into the international computing grid. The national infrastructure is based on middleware from the LHC Computing Grid (LCG) project [1]. LCG provides a common software distribution and site configuration to ensure interoperability between widely distributed sites. LCFGng [2] allows network installation of nodes according to configuration profiles stored on an install server.

Grid-Ireland currently comprises an Operations Centre based at Trinity College Dublin (TCD) and six nationwide sites. The Operations Centre provides top-level services (resource broker, replica management, virtual organisation management, etc.) to all sites. Each site hosts a Grid access gateway and a number of worker nodes that provide compute resources. We aim to make Grid services accessible to a far higher proportion of Irish research institutions in the

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 761–770, 2005.

near future. To achieve this goal we must ensure that the hardware and personnel costs necessary to connect a new site to the Grid are not prohibitive. A standard LCG gateway configuration makes significant hardware demands of a site. A minimum of four dedicated machines are normally required: an install server, providing a configuration and software repository for all nodes; a computing element (CE), providing scheduled access to compute nodes; a storage element (SE), providing data management, and a user interface (UI) providing for job submissions from users.

We have already deployed VM technology at six sites: all UIs are running as User-Mode Linux (UML) VMs on the SE server. We now wish to deploy to eleven new sites, and based on positive experience with the use of VMs, we are currently developing a software distribution that will allow us to run all gateway services for a site on a single computer. This machine will host a number of VMs acting as logical servers, with each VM running its own OS instance. As each VM will appear to be a real machine (both to the server software and to users), there will be no need for special configuration relative to the existing gateways.

1.2 Aims

We aim to make Grid deployment more cost-effective by using VM technology to reduce the number of machines that need to be dedicated to a Grid gateway. We will also develop tools that allow administrators to automate the initialisation and configuration of the VMs, thus simplifying the installation process. We also aim to limit the divergence from a standard Grid site configuration so that the same configuration data (LCFG profiles, software package lists) can be used for all gateways, whether they use VMs or not. We also provide for central management of remote sites: each of the servers must be accessible via both console redirection and Secure Shell (`ssh`). Finally we must provide a simple installation process that can be performed remotely.

1.3 Outline

In the remainder of this paper we describe the advantages of using VM technology to build and deploy Grid gateway services on remote sites. In Section 2 we discuss the factors determining the choice of a good VM system, in Section 3 we briefly describe the architecture of a Grid gateway built on VMs, and in Section 4 we describe the tools we have developed to aid deployment. In Section 5 we outline the installation procedure used to roll out new sites, and in Section 6 we make some observations based on our experience of deploying VM technology. Section 7 discusses related work, and finally Section 8 summarises our findings.

2 Choosing a VM System

Making a good choice of VM technology is crucial to building a secure, fast system that is easy to manage. We briefly describe a range of currently available

VM systems, and choose representative VM systems for evaluation. We also outline the technical and administrative requirements demanded by the task of Grid deployment.

2.1 Overview of VMMs

A virtual machine system provides each user with a complete OS environment tailored to his applications and isolated from other users of the computer. The virtual machines are controlled by a monitor (VMM), which enforces protection and provides communication channels. In the past, VM technology was most widely applied in mainframe computing, for example in IBM's VM/370 system [3]), where it was used to allow many users to share the resources of a single large computer.

Recently, interest has grown in implementing VMMs on commodity hardware and the past few years have seen a stream of commercial and open-source VMMs which provide varying levels of virtualisation. Full virtualisation virtualises a complete instruction-set architecture: any OS that will run on the underlying hardware will run on the VM. Examples include VMWare [4], a commercial product that provides full x86 virtualisation on both Windows and Linux, and Qemu [5], an open-source x86 emulator. Para-virtualisation presents a modified interface to guest OSes, which must be ported to the new VM "architecture"; this approach results in performance close to that of the bare hardware. Xen [6] is a para-virtualised VMM which supports Linux and BSD-based guest OSes. Finally, system call virtualisation provides an application binary interface that enables guest OSes to run as user-space processes. User Mode Linux (UML) [7] is a port of the Linux kernel to run in user-space; it can be run on an unmodified host OS although kernel modifications are available that improve performance.

Our evaluation has focussed on Xen and UML. These are both open-source projects, allowing us to customise the code if we need to. They are also stable projects with active user communities to provide support. Both Xen and UML are compatible with the LCG software: LCFG profiles can be modified to selectively install custom kernels on the correct machines. UML VMs can be run on an unmodified Linux kernel, although specially patched kernels may be used to improve performance. We have excluded commercial VMMs from our experiments due to cost considerations and licensing restrictions.

2.2 Requirements

We aim to run all gateway services quickly, securely and reliably on a single machine and so require a VMM to provide the following features:

Isolation: In a single-box solution, it is important for the VMM to provide isolation between VMs, so that even a catastrophic OS failure in one VM will not affect the others. (A hardware failure will inevitably affect all hosted OSes, but this risk could be mitigated by providing a backup machine to act as a failover.)

s*Storage:* The logical servers each have different storage requirements but must share a limited set of local disks: the VMM should provide a flexible means of

sharing the available disk space between hosted nodes to reduce the need for tricky repartitioning in the case of file systems filling up.

Resource control: The various servers also have different CPU requirements: the VMM should provide a means for controlling CPU utilisation. For example, to preserve interactive performance on the UI it may be necessary to throttle back the CPU utilisation of the other nodes. It would also be useful to be able to partition other resources such as disk and network bandwidth and physical memory.

Low overhead: The VMM should not impose a high performance overhead or significantly reduce system reliability. This is particularly an issue during I/O intensive operations such as installation/upgrade: while almost all VMMs can run compute-bound code without much of a performance hit, few can efficiently run code that makes intensive use of OS services. As the gateway will host the User Interface, the VMs must provide good interactive response times.

Management support: The VMM should also provide features to facilitate management of VM nodes. Such features typically include access to consoles for each VM, a facility for storing VM configurations, and tools for displaying and controlling VMs' resource usage.

2.3 Performance Evaluation

Detailed performance measurements of Xen, VMWare and UML can be found in the main paper describing Xen [6]. The results show that Xen consistently out-performs the other VM systems tested: by a small factor for computation-intensive applications, and by a very large factor for applications using I/O and other OS services.

We have also performed our own measurements of Grid applications; full results may be found in [8]. Figure 1 shows the outcome of tests recording the duration of a first-phase LCFG installation of a UI node — a procedure including the creation of filesystems and the installation of the Red Hat 7.3 OS and LCG middleware (over 700 RPM packages). This procedure takes around seven minutes on a Xen VM, but over an hour on UML. The results of this performance evaluation have led us to migrate our existing UML-based VMs over to Xen.

3 System Structure

Figure 2 shows the structure of our main VM-based application: a single-computer Grid gateway. Each server runs in its own OS on a separate virtual machine: the LCFG install server runs on the first VM and the other servers (CE, SE, UI and WN) run in VMs with filesystems hosted in loopback files on the install server's file system. Xen provides a virtual network interface for each of the VMs. These are bridged onto the real Ethernet card using standard Linux utilities, providing direct network connections. Further details may be found in [8].

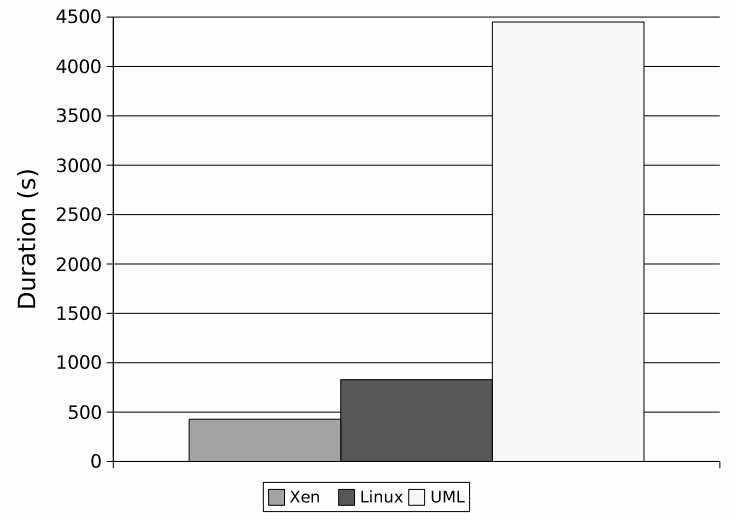


Fig. 1. LCFG installation of UI node

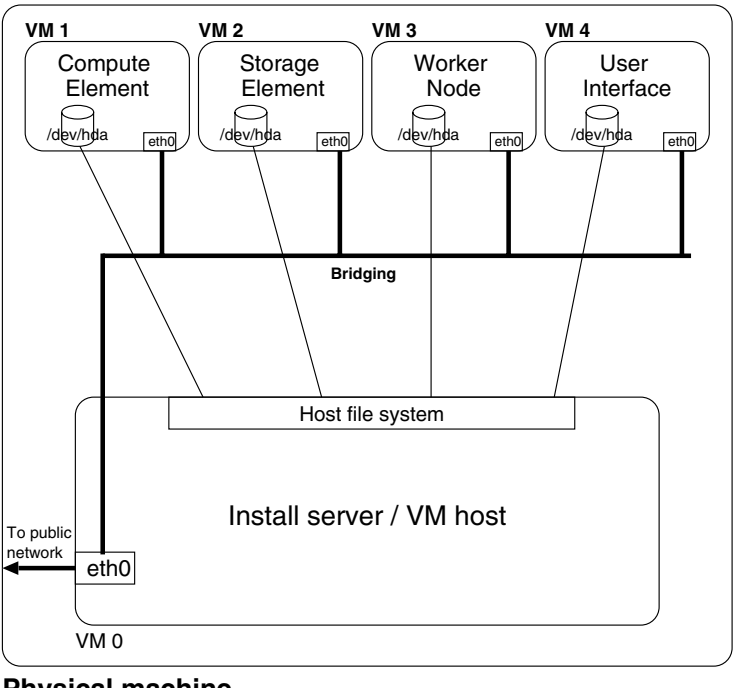


Fig. 2. Architecture of grid gateway

4 Tool Support

4.1 Control Tools

If VM software is to be deployed across multiple sites, we need to make it simple for system administrators to control its operation. This implies that they should not need need detailed knowledge of the VM configuration parameters. For this reason, it is important to integrate the VM system with standard system service control tools. This is most easily done by writing scripts to wrap the tools provided by the VM software.

We have already developed control tools for our UML-based VMs which have been deployed for some time now on the national infrastructure. There are two main components: a service control script used for integrating the VM system with standard boot service configuration, and a command-line tool designed for run-time control of VMs.

The service control script allows for all configured VMs to be automatically started or stopped in a particular run-level. A list of VMs is stored in a file and for each VM, a configuration file contains network settings stored as simple shell variables. The control script reads the configuration variables and creates an appropriate command line for the VM control system.

The principle behind the development of these tools is to provide a level of abstraction that allows administrators to control the gateway services in the same manner as they would other standard services. Configuration settings can be set using standard shell variables: the administrator does not need detailed knowledge of the VM command line parameters.

4.2 Remote Management

There are typically two levels at which remote management must be provided for VMs: access to the machine itself at a low-level, and access to each of the VMs. The first is typically provided by dedicated hardware in the server machines which enables management features such as remote console access over a dedicated serial or Ethernet connection and console redirection allowing access to BIOS settings. The second must be provided by the VM system itself. UML VMs are in fact standard Linux processes, output appears directly on the console where the VM was created. However, when multiple VMs are running, it is convenient to be able to attach to the console later on. We start each VM within an instance of the `screen` utility and give them an unique name that allows us to reconnect later on as necessary. Xen provides its own tools for console-level access: the complete boot sequence for each VM can be observed via a command-line application which can be attached or re-attached at any time.

5 Installation Procedure

In order to deploy software to eleven new sites around the country, it was important to develop a simple installation procedure. We took a three-stage approach:

firstly we installed a base system using a combination of manual installation and the LCFG install tool and took an image of that system; subsequently we copy that image to the target machines' disks, and reconfigure the new servers to prepare them for installation at a specific site.

5.1 Stage 1: Installation of Base Image

The base OS (Red Hat Linux 7.3) was installed from CD, and then the LCFG server software was installed according to the standard procedure. At this stage, we upgraded a number of packages to provide the versions of software required by the Xen tools. We then installed a custom RPM package which included Xen kernels compiled with support for the hardware on the target machine. Changes were made to the LCFG configuration to support the Grid-Ireland layout, and profiles for the various nodes to be hosted were retrieved from a CVS repository.

The next step was to install the various nodes using LCFG. This involved various modifications to the standard LCFG install procedure due to the fact that these were virtual machines rather than real physical machines. Normally, LCFG-installed machines network boot using PXE: this isn't necessary with VMs, as the VMs are started from the command line. Also, network settings can be specified directly, eliminating the need for DHCP. Once the necessary modifications were made to the LCFG install script, the VMs were booted with a root file-system set to the LCFG installation directory, and executed this script as `init`.

The LCFG installation process installs the Red Hat Linux 7.3 operating system and the packages making up the LCG middleware: a total of 700-800 packages depending on the configuration. The first phase of the installation process takes around 7 minutes on a Dell PowerEdge 1750 2.4 GHz machine running Xen. Once all nodes (CE, SE, UI and WN) had been successfully installed, we ran tests to confirm that they were operating correctly. We then shut down the virtual machines and dumped the complete filesystem to a compressed image file, which was approximately 6.6 GB in size.

5.2 Stage 2: Imaging Target Machines

This image file is stored on a portable hard drive, which we then use to install the target machines. We use the standard `dump` format so the file system can be restored to any disk of a sufficient size.

5.3 Stage 3: Configuration of New Servers

The system image transferred to the target machine contains many settings that refer to the original installation, and these have to be updated to reflect the desired configuration. Ideally, this would be simply a case of pointing each individual virtual machine to its new profile, and then allowing LCFG to reconfigure the system. In practice, we have found it simpler to perform some pre-configuration on the new file system before starting the boot process.

The basic steps needed are to update the network settings, to copy the new LCFG profile and to compile it. In the host OS, we mount the filesystem and edit the network configuration files to reflect the new identity of the machine. We then copy the XML version of the profile to the correct location on the VM filesystem. We use the `chroot` program to run the profile compiler within the VM filesystem so that the compiled profile ends up in the right place. All these steps are scripted so that they can be included in an automated process.

When the virtual machine is booted, the LCFG client reads the new profile and performs any reconfiguration necessary. Extensive changes should not be necessary as the main differences between server installations at different sites are the network settings, which we have already modified by this stage. Once the new VM is up and running, a small number of manual steps still need to be performed: this is because LCFG objects have not been written for all system components.

5.4 Deployment to Sites

All previous steps can be carried out at the Operations Centre (subject to site managers providing the necessary network settings for their site). The actual procedure of installation at the site is intended to be straightforward: the main task is to provide network and power connections for the gateway machine.

6 Experience with Virtual Machines

We have been using UML for the past year to share a single machine between an SE and a UI; this configuration currently runs on six sites nationwide. As a result of performance evaluations [8], we are switching to Xen for the next tranche of gateway rollout. The performance overhead due to UML, while just about acceptable for use with a single VM, is too high for our target of five VMs per computer. UML is around ten times slower than Xen for OS-intensive tasks, making node installations and upgrades lengthy processes. We have also found Xen to be more responsive than UML during interactive use.

There are also management benefits: as the VMs' file systems are stored as regular files on the host, we can easily back up an entire site gateway by dumping the host file system. VMs also ease site installation as only a single machine needs to be provided with network connection and power. Installation of individual servers is also more manageable. Even with network installation, unforeseen issues often arise that require physical access to the machines. With VMs this doesn't arise: once the host is up and running, all servers can be easily installed and accessed from the command line.

Full remote console access is more easily arranged with VMs than with real machines. With virtual machines, BIOS settings are simply not an issue (there is no BIOS!), and both Xen and UML provide full console redirection that allows the entire boot process to be monitored.

We have also found that the use of VMs greatly eases the task of reconfiguring existing sites. For example, we recently decided to deploy a test worker node at

each site to allow administrators to run comprehensive tests even when the site's physical compute resources were inaccessible. We decided to run this new node on a VM, and this made the installation procedure very straightforward. We simply took an filesystem image (actually of a UI system — the process would be even easier using a WN image), copied it to a new machine, booted a new VM from this image, and then reconfigured the VM with a new profile and network settings. These operations were all performed remotely over SSH without any need for physical access to the site.

7 Related Work

The work of Figueirdo *et al* [9] is complementary: they propose the use of virtual machines for Grid worker nodes whereas we use VMs for the gateway servers. They aim to support a variety of guest operating systems and so choose a VMM that supports full virtualisation. Other sites within the LCG collaboration have explored the use of VMs: the London e-Science Centre have used UML to provide an LCG-compatible environment on existing cluster machines [10], and Forschungszentrum Karlsruhe have used UML to host their install server [11]. To our knowledge, no-one else has implemented a complete site gateway using VMs. Outside the Grid community, the XenoServer [12] and Denali [13] projects both use VM techniques to support dynamically instantiated application environments for remote users.

8 Conclusion

The use of virtual machines is central to the speedy deployment of new sites in the Grid-Ireland architecture. We have found that using VMs reduces hardware costs, enabling more sites to be deployed for a given aggregate budget and in a given time. This allows a significant increase in Grid participation.

Although we have demonstrated that it is feasible to construct a single-machine Grid gateway using virtual machines, our experiments show that the choice of VM technology is crucial. User-Mode Linux, while in widespread use, is impractical for our purposes due to its extremely high overhead for OS-intensive tasks. Xen, in contrast, performs well across a range of applications. Because Xen provides an OS environment that is indistinguishable from a regular OS instance, software servers can be run with the same configuration as on dedicated machines.

VMs also speed up deployment as an entire gateway configuration can be quickly imaged onto a single target computer. Remote management is also easier with VMs: full console access is available without the need for specialised hardware and software.

We have already experienced significant benefits by using VM technology in our site installations. We believe that this approach will be of interest to many sites wishing to connect to the Grid.

Acknowledgements

We would like to thank the Xen team at the University of Cambridge for developing the Xen VMM and for providing useful support. Dell Ireland kindly donated the hardware for the VM-based gateways. The original UML work drew on a configuration by David Coulson [14].

References

1. LCG: LHC Computing Grid Project (LCG) home page. `http://lcg.web.cern.ch/ LCG` (2004)
2. Anderson, P., Scobie, A.: LCFG — the Next Generation. In: UKUUG Winter Conference, UKUUG (2002)
3. Gum, P.H.: System/370 Extended Architecture: Facilities for Virtual Machines. IBM Journal of Research and Development **27** (1983) 530–544
4. Devine, S., Bugnion, E., Rosenblum, M.: Virtualization system including a virtual machine monitor for a computer with a segmented architecture. US Patent (1998)
5. Bellard, F.: QEMU CPU emulator. `http://fabrice.bellard.free.fr/qemu` (2004)
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, ACM (2003)
7. Dike, J.: A user-mode port of the Linux kernel. In: Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta, USENIX (2000)
8. Childs, S., Coghlan, B., O'Callaghan, D., Walsh, J., Quigley, G.: A single-computer grid gateway using virtual machines. In: Proceedings of the IEEE Conference on Advanced Information Networking and Applications (to appear). (2005)
9. Figueiredo, R.J., Dinda, P.A., Fortes, J.A.B.: A Case for Grid Computing on Virtual Machines. In: Proceedings of the International Conference on Distributed Computing Systems. (2003)
10. McBride, D.: Deploying LCG in User Mode Linux. (`http://www.doc.ic.ac.uk/~dwm99/LCG/LCG-in-UML.html`)
11. Garcia, A., Hardt, M.: User Mode Linux LCFGng server. `http://gridportal.fzk.de/websites/crossgrid/site-fzk/UML-LCFG.txt` (2004)
12. Fraser, K.A., Hand, S.M., Harris, T.L., Leslie, I.M., Pratt, I.A.: The Xenoserver computing infrastructure. Technical Report UCAM-CL-TR-552, University of Cambridge Computer Laboratory (2003)
13. Whitaker, A., Shaw, M., Gribble, S.: Denali: Lightweight virtual machines for distributed and networked applications. In: Proceedings of the USENIX Annual Technical Conference. (2002)
14. Coulson, D.: UML-based pseudo-dedicated hosting service. (`http://uml.openconsultancy.com`)

Development of Cactus Driver for CFD Analyses in the Grid Computing Environment

Soon-Heum Ko[1], Kum Won Cho[2], Young Duk Song[2], Young Gyun Kim[3], Jeong-su Na[2], and Chongam Kim[1]

[1] School of Mechanical and Aerospace Eng., Seoul National University
[2] Dept. of Supercomputing Application Technology, KISTI
ckw@kisti.re.kr
[3] School of Computer Eng., Kumoh National Institute of Technology

Abstract. The Grid Computing[1] has been paid much attention from researchers as an alternative to parallel computing for its unlimited number of potential resources available and as an easier way to build collaborative environments among multiple disciplines. However, the difficulty in establishing the environments and accessing and utilizing the resources has prevented application scientists from using Grid computing. Thus, the present study focuses on building PSE(Problem Solving Environment) which assists application researchers to easily access and utilize the Grid. The Cactus toolkit, originally developed by astrophysicists, is used as a base frame for Grid PSE. Some modules are newly developed and modified for CFD(Computational Fluid Dynamics) analysis. Simultaneously, a web portal, Grid-One portal, is built for remote monitoring/control and job migration. Cactus frame through the web portal service has been applied to various CFD problems, demonstrating that the developed PSE is valuable for large-scaled applications on the Grid.

1 Introduction

Many researches have been conducted on developing user-friendly interfaces for application scientists to easily utilize advanced computing environment, i.e., the Grid computing environment. A specialized computer system which offers computational conveniences required for a specific problem is called PSE(Problem Solving Environment) and, nowadays Nimrod/G[2], Triana[3], and Cactus[4] are applied briskly. Of the PSEs, Cactus was firstly developed for collaboration among astrophysicists, but it is applicable to the CFD(Computational Fluid Dynamics) analysis. Users can easily attach their own application solver, a toolkit for checkpointing, a toolkit for parallel I/O, a visualzation software, and so on, to Cactus flesh which is an integration of multiple computational techniques. Additionally, Cactus can operate in various computer platforms including a single machine, parallel clusters, super computers, and so on. And, application researchers can utilize many softwares and libraries inside the Cactus frame, like Globus toolkit, HDF5 I/O, PETSc library, and visualization tools.

Present research focuses on improving Cactus frame for CFD analysis. So, a CFD flow solver is appended to Cactus frame and modules for general coordinate I/O are

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 771 – 777, 2005.

newly developed. Additionally, researches on developing computational toolkits including advanced visualization, job migration and load balancing, and portal service, are conducted by computer scientists. And then, a lot of validations are carried out utilizing Cactus-based CFD analyzer through a portal service and job migration test is conducted on K* Grid. From these studies, CFD flow solver is successfully implemented to the existing Cactus frame and the collaboration between computer scientists and application researchers is accomplished.

2 Cactus and CFD

To apply Cactus frame to CFD analysis, CFD researchers should implement their flow solver into Cactus frame. For application scientists, implementing their flow solver al-ready developed into a new frame may seem absurd at a glance. Thus, the advantages and disadvantages of applying Cactus to CFD analysis are to be investigated in the present paragraph.

The advantages of Cactus-based CFD analysis are as follows. Firstly, application scientists can perform the Grid computing without profound knowledge of the Grid, in-creasing the problem size. Additionally, as all the researchers modularize their application solver, the collaborative research can be easily accomplished. And, the most advanced computer science technologies can be achieved without associated knowledge by utilizing Cactus modules provided by computer scientists.

However, when researchers analyze small-scaled problems or develop numerical techniques, Cactus is not necessary as analysis can be conducted inside a local organization. Rather, it can have a bad influence as learning how to use Cactus and implementing their flow solver into Cactus frame requires additional time. And, implementation process is somewhat complex as flow solver should be generalized while each flow solver is developed for the analyses of specific problems. Finally, as CFD researches by Cactus are not activated until now, supports for CFD analysis, like mesh generation using body-fitted coordinate and visualization of CFD result, are not sufficient.

3 Cactus-Based CFD Analysis

3.1 3-D Inviscid Flow Analysis Using Cactus

The procedure of 3-D inviscid flow analysis is as follows. At the pre-processing level, mesh generation is accomplished and at initial level, initialization of flow conditions, inputting mesh points and transformation to curvilinear coordinate are performed. And then, at iteration step, determination of time scale by local time stepping method, flux calculation by spatial discretization, time integration and application of boundary conditions are accomplished. Then, at the post-processing level, resultant data are visualized and analyzed. In present research, each component comprehended in the analysis process is made as a separate thorn of Cactus frame. Figure 1 shows the flow chart of 3-D inviscid flow analysis after converting each subroutine to thorn and scheduling thorns to the time bins of Cactus.

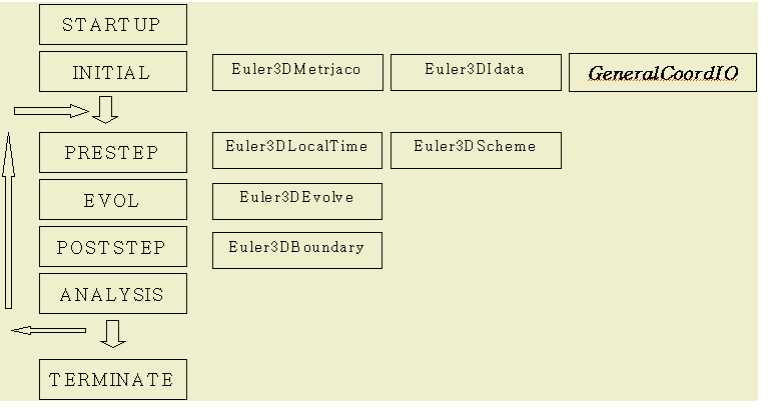


Fig. 1. Time Bins and Thorns for 3-D Inviscid Analysis

Each thorn is configured by CCL(Cactus Configuration Language) scripts and, analysis subroutine of flow solver is stored inside the src/ directory with the modification of parameters and functions to Cactus format.

3.2 Numerical Results

The flowfield around a wing is analyzed by using Cactus-based CFD analyzer. Firstly, to confirm the possibility of CFD analysis using body-fitted coordinates, RAE-2822 airfoil is analyzed on single processor. The present configuration is basically 2-dimensional, but the mesh is extended to z-direction, making a 3-D mesh with symmetric condition along the z-direction. Total mesh points are 161×41×3. Mach number is set to be 2.0 and 0.78, depending on flow conditions.

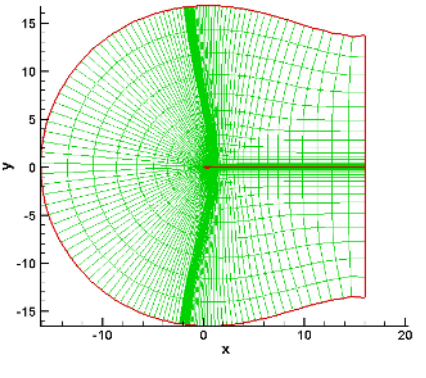


Fig. 2. RAE-2822 Mesh

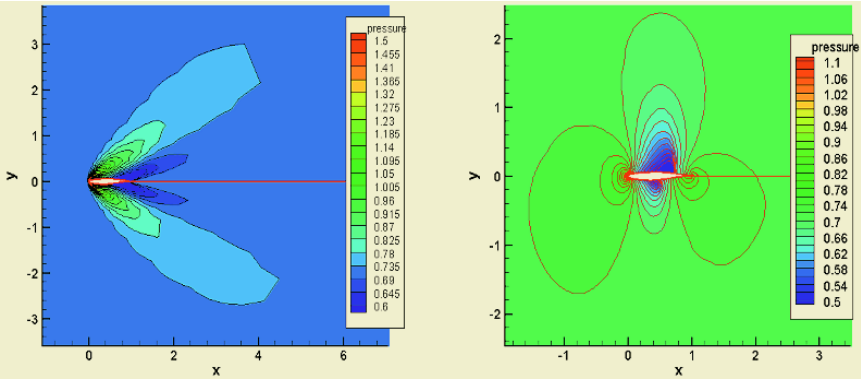


Fig. 3. Supersonic(L) and Transonic(R) Analysis

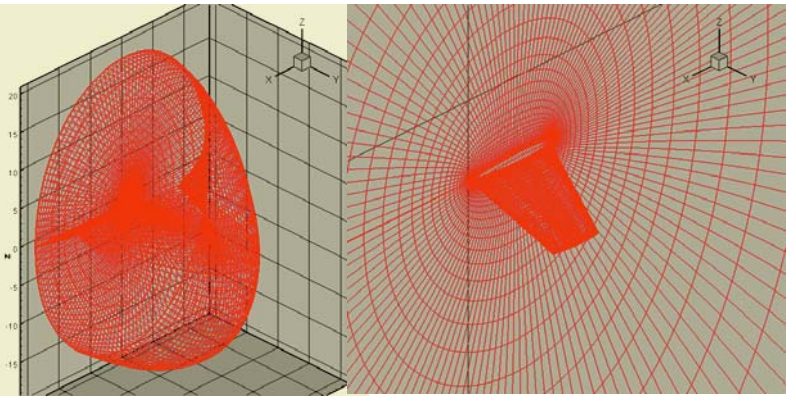


Fig. 4. 3-D Mesh around Onera-M6 Wing(O-type)

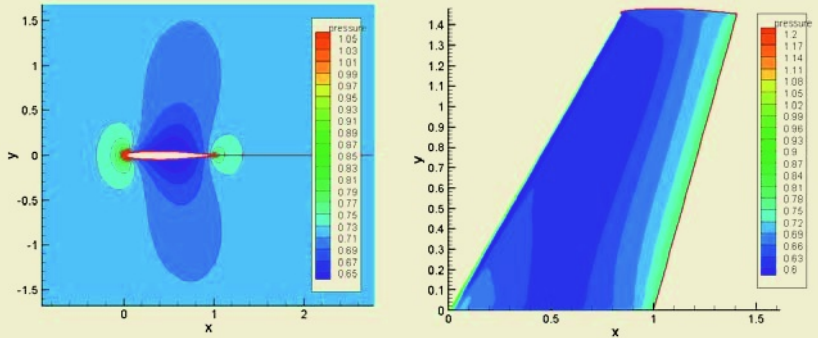


Fig. 5. Pressure Contours along Root of the Wing and Wing Surface

As Cactus CFD solver is proved to be valid from the analysis of RAE-2822 airfoil, the flowfield around a 3-D Onera-M6 wing is analyzed by using K* Grid, a Korean

Grid testbed. The mesh system is shown in figure 4 and total mesh points are 141×33×65. 6 processors are co-worked and, Cactus frame automatically partitions the whole domain by 1×2×3 zones. Mach number is 0.78 and angle of attack is 0 degrees. And, the resultant pressure contour is shown in figure 5. Pressure increase near the leading edge of the wing and gradual decrease of pressure shows the correctness of the flow analysis. However, when more than 8 processors are utilized, periodic boundary condition should be applied along the tangential direction of wing surface and the result is not correct when periodic boundary condition is applied. Thus, the improvement of periodic condition is required.

3.3 Utilization of Grid-One Portal Service

As is mentioned, utilizing the Grid services is very hard to application scientists. Thus, a Grid-One portal is built and Cactus-based CFD analysis is conducted through the portal service. The frame of a portal service is built on the basis of GridSphere[5], an open-source portlet framework developed as a part of the GridLab project[6].

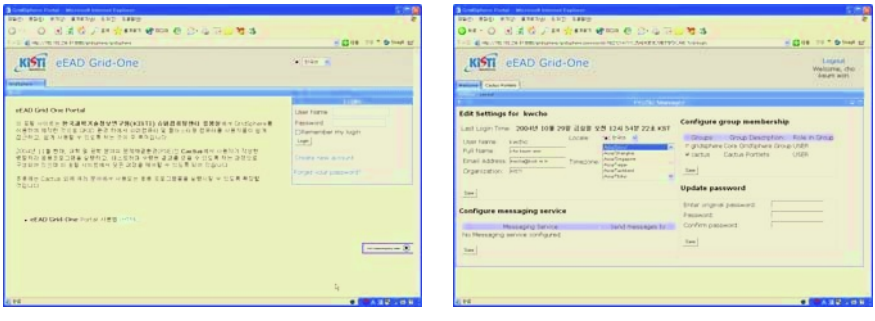


Fig. 6. Log-in Process of Grid-One(L) / Compile and Configuration Process(R)

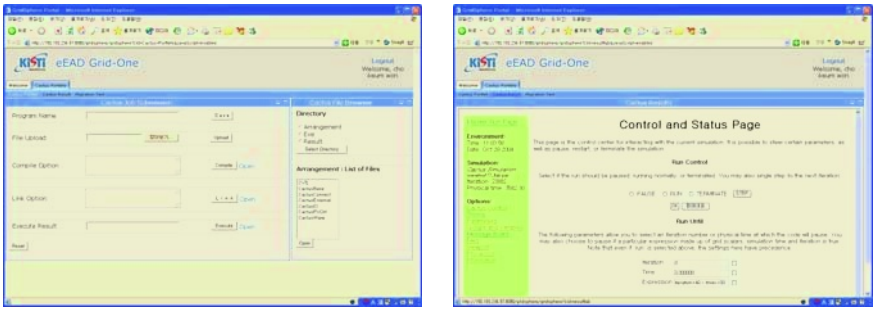


Fig. 7. Cactus Mode of Grid-One(L) / Cactus Control Page(R)

The usage of Grid-One portal is as follows. At first, to get the service of Grid-One, users should register to the Grid-One portal and log-in. After logging-in, users will configure their own information like e-mail, password, and application part, and

check 'Cactus' in the configure group membership item to start Cactus-based analysis. After clicking 'Save' button, Cactus mode is shown in the browser. At Cactus mode, users will upload their own Cactus-based application modules and then, compile and link uploaded Cactus program. In ‘compile’ and ‘link’, users can handle the compiling and linking options(if you are an advanced user) and, if users don’t add any options, the portal will automatically compile as a serial program. During and after the compiling and linking processes, users can confirm the result of compiling process by clicking ‘open’ button. After linking is complete, users start their analysis program. During analysis, on-going processes of the application problems are simulated via web browser and real-time control is possible.

Cactus CFD simulation test with job migration is accomplished through Grid-One portal as shown in figure 8. Firstly, Cactus CFD solver is uploaded to the portal server, located in the K* Grid gateway. After compiling, CFD analysis starts in the gateway server. After 10 minutes of analysis in the gateway, job migration manager finds the better resources and analysis file with requested data files migrate to Venus cluster, one of computing resources in K* Grid. Migrated solver automatically restarts from where the solver ended in former resources(in the gateway). And, after 10 minutes, migration manager searches for better resources again. At that time, finding better re-source inside testbed, migration manager transfers required files to the resources in Konkuk university, Seoul. After 10 minutes in Konkuk university, flow solver completes its analysis.

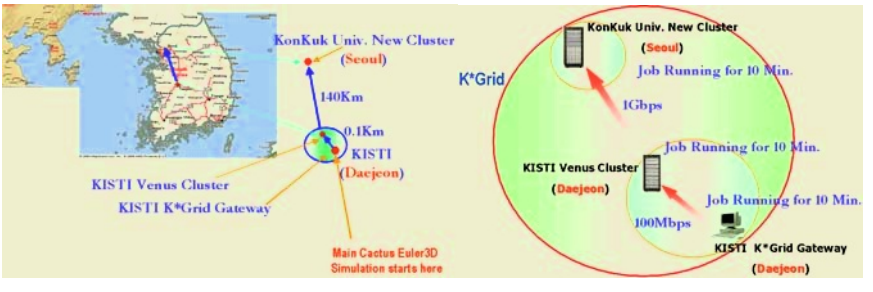


Fig. 8. Job Migration Scenario on K* Grid

From the users’ viewpoint, they only have to upload their analysis program to portal server and compile inside the portal. During analysis, migration manager automatically searches for a better resource and transfers the solver to the optimal resource in cooperation with the portal service. As is seen in figure 9, each resource in the Grid testbed starts migration manager when the analysis starts in the portal server and status report of each resource is transmitted to portal server. Based on the reports, migration manager decides the optimal resource for that analysis and, the result is printed in the web page of portal service. By looking at the messages printed in the web browser, users can get the information about the current job status and the location. And, after the analysis is complete, resultant data are automatically transmitted to portal server, enabling downloading the data file and analyzing the result.

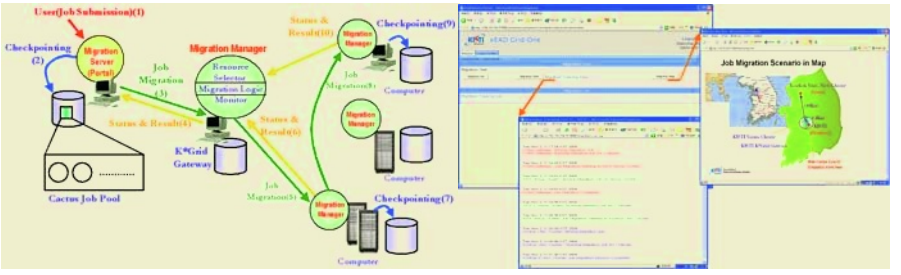


Fig. 9. Procedure of Job Migration

4 Concluding Remarks

The present research applied Cactus PSE to the analysis of CFD problems. From the understandings of the structure of Cactus, researchers developed CactusEuler3D arrangement with General Coordinate I/O routine that can utilize body-fitted coordinate in the Cactus frame. And Cactus-based CFD solver is validated by the analysis of the flowfield around the wing. From these researches, Cactus PSE is shown to be applicable to CFD analysis and the possibility of multi-disciplinary collaboration by Cactus frame is presented.

Simultaneously, Grid-One portal is built for various applications including CFD. The frames are built on the basis of GridSphere and, Cactus CFD analyzer and migration manager are appended to the portal. The application users can easily utilize the Grid service only by clicking the menus in the web browser. During the analysis, portal manager offers monitoring and controlling service and migration manager automatically utilizes the optimal usable resources. Developed portal service and migration manager are very helpful to application users in utilizing the Grid resources and they will play a major part in the construction of Korean e-Science infrastructure.

References

1. M. M. Resch, "Metacomputing in High Performance Computing Center," IEEE 0-7659-0771-9/00, pp. 165-172 (2000)
2. D. Abramson, K. Power, L. Kolter, “High performance parametric modelling with Nimrod/G: A killer application for the global Grid,” Proceedings of the International Parallel and Distributed Processing Symposium, Cancun, Mexico, pp. 520–528 (2000)
3. http://www.triana.co.uk/
4. http://www.cactuscode.org/
5. http://www.gridsphere.org/
6. http://www.gridlab.org/

Striped Replication from Multiple Sites in the Grid Environment*

Marek Ciglan, Ondrej Habala, and Ladislav Hluchy

Institute of Informatics, Slovak Academy of Science,
Dubravska cesta 9, 845 07 Bratislava, Slovakia
{marek.ciglan, ondrej.habala, hluchy.ui}@savba.sk

Abstract. Grid technology, as a highly distributed computing environment, requires an optimized access to the data resources to increase data availability. In this paper, we propose a replication technique which is based on parallel transfers from multiple sites containing replicas of the desired file. From each site, we transport in parallel only a portion of the given data source, obtaining the whole file at the end of the process. We describe the work related to the data replication; then we discuss two algorithms for striped replication optimization that aims at the minimization of the time necessary for data transfer. Finally, we present the results of the striped replication mechanism achieved by the prototype implementation of the striped replication algorithm. We compare them with the results of the standard replication tools and show interesting performance improvement.

1 Introduction

Grid computing is an important new concept for distributed processing, based on the idea of globally shared computer resources between organizations, such as disk space, information and computational power [8]. Grid computing helps users, who need to run computational and data intensive tasks, which could be too demanding and time consuming for a single supercomputer or computational cluster.

Such a task is distributed within the grid to several grid nodes, saving execution time. This brings immense computational power for relatively small cost. In such highly distributed environment, an efficient data management is needed to keep track of the data sources in the grid and to optimize the network traffic. Many grid applications request large data collections, which, possibly, have to be transferred via network to the grid nodes that execute given jobs. To minimize the network traffic, transportation of the files, the data replication strategy

* This work is supported by EU 6FP RI(III) project: Enabling Grids for E-sciencE (2004-2006) INFSO-RI-508833, EU 5FP IST RTD project: CROSSGRID Development of Grid Environment for Interactive Applications (2002-05) IST-2001-32243 and the Slovak Scientific Grant Agency within Research Project No. 2/3132/23.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 778–785, 2005.

is used. Data replication means creating copies, replicas, of files that are often required by grid jobs. This leads to having multiple copies of a single file across several grid nodes, which helps to reduce access latencies to the data.

In general, there are two levels of data access optimization: short-term and long-term optimization. Short-term optimization aims at delivering replica of required file in the shortest time possible, given a requesting grid node and logical file name.

Long-term optimization concerns global reduction of network traffic in the grid by deciding which files should be replicated (deleted) on (from) which grid sites. Current state-of-the-art grid data management tools select the file replica with lowest access cost, given a requesting grid node and logical file name of the data source. The selected replica is then transferred to the grid site. We describe work related to data replication in section 2. In this paper, we present an approach to the short-term replica optimization for grids, that has the potential of further acceleration of the replication process, in the case when more than one replica of desired data source is present in the grid. We propose a striped replication mechanism, a method which doesn't transfer the data from a single grid site only, but is rather based on parallel transfers from multiple sites containing replicas of the desired file. From each site, we transport only a portion of the given data source, obtaining the whole file at the end of the process. Doing so in a parallel way, we can expect time reduction of replica creation. However, transferring file stripes from multiple sites can eventually increase the transfer time, if the process isn't supervised and optimized. For example, if one of the file replica resides on the grid site with extremely low connectivity, attempt to transfer the identical potions of file from each site could results in decrease of overall replication speed. Optimization procedure for striped replication management is thus needed. We discuss two optimization algorithms for the striped replication in the grid environment in section 3.

First algorithm computes portions of replicas that will be transferred from distinct grid sites, according to the information about replica's access costs obtained from monitoring services. This is done before the transfer actually begins, the method is static during the transfer period. Second algorithm is more flexible during runtime and can dynamically change the amount of data that is transferred from different sites, according to actual network performance. We outline the reasons why we choose the second algorithm for the prototype implementation of the striped replication tool.

In section 4, we present the results of replication tests obtained by our prototype implementation of striped replication. We compare them to the results obtained by standard replication tools and show interesting performance increase. Finally, we discuss the advantages and disadvantages of our approach.

2 Related Work

A short-term replica optimization service (ROS) for grids was implemented in European Data Grid (EDG) project [3]. Main function of EDG ROS service is

to evaluate a cost for accessing a set of files from grid sites and to choose the cheapest replica to transfer. It uses gridFTP functionality to increase transfer speed by opening multiple TCP streams from a single site.

A lot of attention to the acceleration of the file replication process was given in peer-to-peer (p2p) systems such as BitTorent, Slurpie, Gnutella.

BitTorrent [9] is a file distribution system. Shared files, within this framework, are made available using HTTP server. When multiple users are downloading the same file at the same time, they upload the pieces of the file to each other. System keeps metadata information about each file, containing file name, size, hashing information and url of a tracker. Trackers are services that help file downloaders to find each other. Information from the tracker is used to find other peer and download management is then handled strictly in the interaction between peers. Shared files are logically split into pieces, downloading peers propagate information about file pieces they have available to each other. The 'rarest first' method is used to decide which file's block would be transferred. BitTorent clients download the rarest file's pieces first, leaving more common ones for later. To prevent transfer slow-down, because of slow transfer rates from certain peers, a transfer choking feature was introduced. Peers try to maximize its own download rates by transferring from whoever they can and deciding which peers to upload file pieces. The choking is a temporally refusal to upload file to the given peer. Peer A stops sending blocks to peer B, chokes the connection to B, until B sends A a block or a time out occurs. The choking encourages peers cooperation. Decisions as to which peer to upload and download from are based strictly on current download rate. As calculating current transfer rate meaningfully is a difficult problem (the bandwidth shifts rapidly over time), BitTorent peer change a single connection, performs 'optimistic unchoke', regardless of the current download rate.

Slurpie [10] is a p2p protocol for bulk data transfer, specifically designed to reduce client download times for large files and to reduce load on servers. All nodes downloading the same file contact the topology server and form a random mesh. The nodes in the random mesh propagate progress updates to each other. This information is used to coordinate the data transfer. Slurpie uses an bandwidth estimation technique to make an informed decision about number of connections to keep open, number of edges to keep in mesh and update propagation rate. Considering the download decision, Sulrpie download blocks served by peers before block served by the source server. Sulrpie adapts to varying bandwidth conditions and scale the number of neighbors as the group size increase.

Both p2p and grid systems use replication strategy to increase data availability, however the setting of p2p and grid environments are different. Connections and disconnections of nodes (users) in the p2p system are quite dynamic, grate effort is focused on locating required file's replicas and on discovering which peer has which file blocks. In the current state of grid environment, we can easily determinate locations of file replicas and only complete replicas, containing all blocks, are present. We try to introduce striped replication principle in the scope of available grid services and we focus primary on the problem of data transfer optimization.

3 Striped Replication from Multiple Grid Sites

The striped replication in grid environment is enabled by capability of GridFTP protocol, which allows the access and the transport of only a portion of the whole file from a grid site (site's Storage Element).

The idea behind striped replication is to transfer portions of desired file's replicas from different grid nodes to further increase the speed of replication. Theoretically, if there are two replicas of the given file in the grid, placed on the sites with the same connectivity considering the site where we want to replicate the file and we replicate this data source in the striped way, 50% of the file from one site and complementary 50% of the file from other site, we can achieve 50% acceleration of the transfer comparing to the replication from a single source. The attempt to transfer file in a striped way may potentially result in decreasing the file transportation speed, if the process is not managed properly according to the actual state of the grid system. For example, if one of the file replica resides on the grid site with extremely low connectivity, attempt to transfer the identical potions of file from each site could results in decrease of overall replication speed. In this section, we describe two algorithms for optimization of the striped replication and we compare them.

The first one is a static approach which computes the portions of distinct replicas that will be transferred before the transfer begins. This method use information from optimization service about the transfer speed from involved grid sites. The second algorithm is more dynamic in nature, it starts by trying to transfer even portions of replicas from different sites and in the course of the data transmission it changes dynamically the amount of data for transfer from different sites, according to the actual performance of the process.

3.1 Static Striped Transfer Mechanism

Our task is to replicate a file in a striped way from multiple grid sites and to do this by computing the portions of involved replicas which will be transferred, before replication actually begins. We can formulate this problem in the following way: Let's have replicas $r_1, \ldots, r_n$ of the data source with given GUID. For each r_i, there is a constant b_i which expresses the amount of bytes that can be obtained from given replica per second. The constants b_i are acquired from the replica optimization service and define current connectivity to the replica. Using this information, we want to find the optimal distribution of replicas partitions for distinct r_i, so that the time of the replication is minimized. We use the following equation as a basis of our thoughts:

$$s = t(b_1 + b_2 + \ldots + b_n)$$

Where s is the given file's size, $b_1, \ldots, b_n$ are the byte-per-second constant for each replica and t is the transfer time. This is a simplified and naive approach because:

- Numbers b_i are considered as constants for the single file replication, which doesn't reflect the dynamic, changing nature of network load.

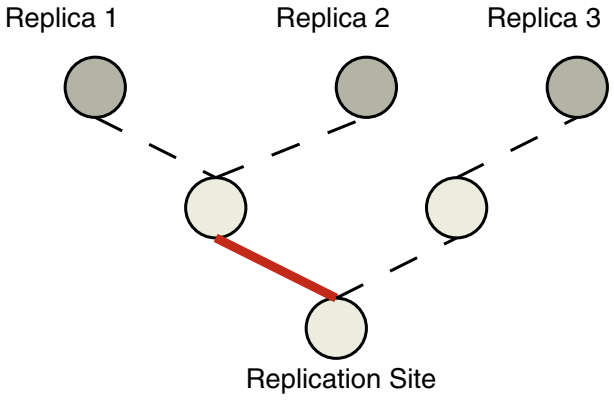


Fig. 1. Shared transfer line for Replica 1 and Replica 2 on the path to Replication Site

- We do not know the exact topology of the network. So if transfer paths from r_i and r_j to the replication site, shares at some point the same line, the transfer speeds b_i, b_j may be eventually lower in reality than those rendered by optimization service, because of available bandwidth of the shared line (figure 1)
- The sum of all b_i can eventually be higher than available bandwidth at local site

More reasons could be found to confirm the naivety of presented approach, still we can use it as a rough approximation. From equation above we can compute the minimal time needed to transfer the file. Now we can simply obtain desired amount of replica portions for all r_i as $p_{r_i} = b_i t$. Doing so, we gather sufficient information to begin the striped replication process.

3.2 Dynamic Striped Transfer Mechanism

In this subsection, we propose the dynamic striped replication mechanism. This method doesn't use the information from the replica optimization service about the connectivity of sites which stores replicas of the desired data source. The only information we have at the beginning of the process, is the location of those replicas.

We begin the striped replication process by initiating n striped transfer threads, each for distinct replica and we try to download even portions of data by all the initiated transfer threads. In the ideal case, this will happen and parallel transfers from replicas grid sites will finish at the same time and the replication time will be $\frac{1}{n}$ of the time required for the replication from a single site.

However, we cannot expect this to happen really often. We propose an optimization technique for dynamic change of the transferring file portions. When some of the parallel striped transfer thread finishes it's replica portion download

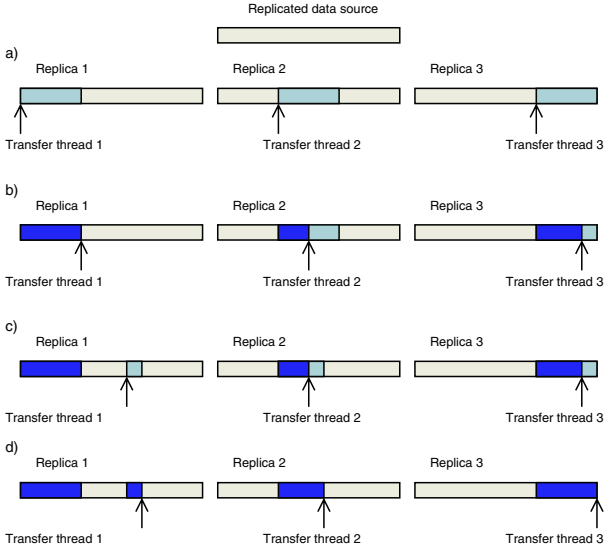


Fig. 2. Illustration of striped replica transfer. Light blue parts symbolize file portions assigned to transfer, dark blue parts sybolize portions already transfered.

a) Situation after the initiation of striped transfer
b) Situation after one of the transfer threads finishes it's assigned portion(s) of replica
c) Situation after reasigning replica portions to transfer threads
d) Possible situation at the end of the process

(and would be idle for the rest of the execution if not handled otherwise), we assign to this thread another portion of the replica, which is equal to the half of the unfinished portion of the transfer thread with largest unfinished amount of data and remove equivalent portion from the latter.

In other words, let there be n replicas of a data source which we need to replicate. We initiate n transfer threads for distinct stripes of the file at different n grid sites and begin transfer. Suppose that thread tr_i finishes transfer of assigned replica stripe and there are other threads that are still transmitting their portions of replicas. Let tr_j be the thread which had transferred the smallest amount of the data. Suppose that tr_j was initiated to replicate portion of the file from the byte p to the byte r and till present, k bytes was downloaded. We split the interval $< p + k, r >$ to two halfs and assign $< p + k, (r\frac{p+k}{2}) >$ to tr_j and $[(r\frac{p+k}{2}), r >$ to thread tr_i.

This process is repeated each time some striped transfer thread finishes it's assigned portion(s) of replica. For better illustration of the process see figure 2. Dynamic striped replication from multiple grid sites reflects the changing status of network load and so, brings necessary flexibility to the replication process. Moreover, this method doesn't need to use information from the replica optimization service.

4 Experimental Results

Intuitively, striped parallel transfer brings important increase of replication speed. The acceleration of replication process clearly depends on the data transport speed from involved replica sites. To give reader the idea of such replication method performance in the real grid environment, we present in this section our experimental results.

We have implemented the prototype of striped replication from multiple grid sites and tested its performance in grid environment. The implementation is made in Java programing language with use of CoG 1.2 API libraries - developed by Globus Alliance [6]. The dynamic striped replication strategy was used in the implementation. We used EDG replica manager tool as a reference implementation of grid data replication, to compare obtained results.

Before presenting averages of the replication time savings during tests, we describe one motivation test case. We replicated a file of 223.9Mb size [1]. There were two replicas of given data source in the testbed. Transfer time of the best replica to the local node using EDG replica manager tool took 713 seconds, using one tcp stream from the site which stored the cheapest replica (the replica with the lowest access cost with the respect to the replication site). The transfer time of striped replication from both gird sites containing given file's replica took 405 seconds (using 1 tcp stream from each site), performing 43% time saving for data transfer. Then we replicated the file to third grid site and run the striped replication from three sites. This transfer took 209 seconds, accomplishing 71% time saving comparing to previously done EDG replica manager transfer. We performed test with two and three replicas of the replicated data source in the grid, obtaining in average 37% time saving for two and 55% for three replicas (compared to the transfer time of the replica from a single site which was selected as the cheapest by EDG replica manager tool).

Concerning disadvantages, striped replication isn't very useful for small data sets. Another disadvantage is that, as the file is transferred in smaller portions, data stripes have to be joined to the single file after the transfer is completed. This requires additional time at the local site.

The advantages of striped replication approach are:

- important acceleration of replication process
- distribution of network load
- method doesn't use the replica optimization service, monitoring services.

5 Future Work

We plan to integrate described mechanism with Replica Location Service, to obtain list of data source's replicas automatically (providing only data source's

[1] The replicated file was actually a short video presenting Data Grid project; we used it as a kind of tribute to the fine work done in this project.

LFN or GUID) and to implement striped replication from multiple grid sites as a web service.

6 Summary

In this paper we have presented the method of striped replication from multiple sources in the grid environment. We have proposed two optimization algorithms for the striped replication. Finally we have presented experimental results of the tests of the prototype implementation of striped replication, performed in the grid environment. The results of tests are promising, they show important time savings compared to currently used method.

References

1. Chervenak A., Deelman E., Foster I., Guy L., Hoschek W., Iamnitchi A., Kesselman C., Kunszt P., Ripeanu M., Schwartzkopf B., Stocking H., Stockinger K., Tierney B., Giggle: A Framework for Constructing Scalable Replica Location Services, Proceedings of SC2002 Conference, November 2002.
2. The Globus Project, www.globus.org
3. EU Data Grid project, WP2, replication, RLS, http://edg-wp2.web.cern.ch/edg-wp2/replication/
4. EU Data Grid project, WP2, optimization, ROS, http://edg-wp2.web.cern.ch/edg-wp2/optimization/ros.html
5. Kunszt P., Laure E., Stockinger H., Stockinger K., Advanced replica management with Reptor, 5th international conference on parallel processing and applied mathemetics, 2003
6. Commodity Grid Kits, http://www-unix.globus.org/cog/
7. Manohar M., Chervenak A., Clifford B., Kesselmann C., A Replica Location Grid Service Implementation, GGF 10 Workshop, 2004
8. I. Foster and C. Kesselman. Computational Grids. The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufman, 1999
9. Cohen B., Incentives Build Robustness in BitTorrent, http://bittorrent.com
10. Sherwood R., Braud R., Bhattacharjee B., Slurpie: A Cooperative Bulk Data Transfer Protocol, Proceedings of IEEE INFOCOM, March 2004

The Gridkit Distributed Resource Management Framework

Wei Cai, Geoff Coulson, Paul Grace, Gordon Blair,
Laurent Mathy, and Wai-Kit Yeung

Computing Department, Lancaster University, UK
{w.cai, geoff, p.grace, gordon, laurent,
yeungwk}@comp.lancs.ac.uk

Abstract. Traditionally, distributed resource management/ scheduling systems for the Grid (e.g. Globus/ GRAM/ Condor-G) have tended to deal with *coarse-grained* and *concrete* resource types (e.g. compute nodes and disks), to be *statically configured and non-extensible*, and to be *non-adaptive* at runtime. In this paper, we present a new resource management framework that tries to overcome these limitations. The framework, which is part of our 'Gridkit' middleware platform, uniformly accommodates an extensible set of resource types that may be both *fine-grained* (such as threads and TCP/IP connections), and *abstract* (i.e. represent application-level concepts such as matrix containers). In addition, it is *highly configurable and extensible* in terms of pluggable strategies, and supports flexible *runtime adaptation* to fluctuating application demand and resource availability. As a key contribution, the notion of *tasks* enables resource requirements to be expressed orthogonally to the structure of the application, allowing intuitive application-level QoS/ resource specification, highly flexible mappings of applications to available distributed infrastructures, and also facilitates autonomic adaptation.

1 Introduction

The task of a Grid resource management/ scheduling system is to appropriately map a set of distributed applications onto an underlying Grid infrastructure that consists of a diverse set of interconnected computational nodes. This is not an easy task if application demands are to be met without wasting resources. Ideally, applications should share nodes to maximise the exploitation of scarce resources, and the environment should dynamically adapt the execution of applications to reflect fluctuating runtime application demand and resource availability.

Presently, the major player in Grid resource management is Globus/ GRAM [10], which employs an architecture consisting of 'brokers' and 'co-allocators'. Brokers map high-level resource requirements to concrete requirements. They also locate suitable computational nodes. Co-allocators then allocate these nodes, and initiate the execution of appropriate parts of the application on them.

While this scheme is in successful use, it has a number of limitations. First, it deals only with *coarse-grained* resource specifications (e.g. whole machines or at best

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 786 – 795, 2005.

processes) rather than fine-grained resources like threads, buffer pools, or TCP/IP connections. Second, it deals only with *concrete* resource specification and doesn't support abstract resources that are meaningful to particular applications (e.g. matrix containers or EDF schedulers). Third, it is *statically-configured and non-extensible*—i.e., its behaviour in many dimensions can only be changed—if at all—by restarting the system. Finally, it is *non-adaptive*—i.e., the resources allocated at application launch-time cannot be adjusted at runtime.

These limitations are not peculiar to Globus/ GRAM—they are shared, at least in part, by most other distributed resource management proposals (see section 4). In this paper we present a new resource management framework that attempts to address these limitations and thereby improve both application-level flexibility and infrastructure-level resource exploitation. The remainder of the paper is structured as follows. Section 2 provides brief background on the 'Gridkit' middleware platform in which our framework is embedded. Subsequently, section 3 offers a detailed description of the framework itself, section 4 discusses related work, and section 5 concludes.

2 Background on the Gridkit Middleware Platform

The Gridkit middleware platform [11] is intended to provide flexible high-level support to complex and 'advanced' Grid applications such as forest fire management and environmental informatics systems. As well as traditional Grid functionality, such applications involve elements such as sensor network infrastructure, mobility, and collaborative visualisation.

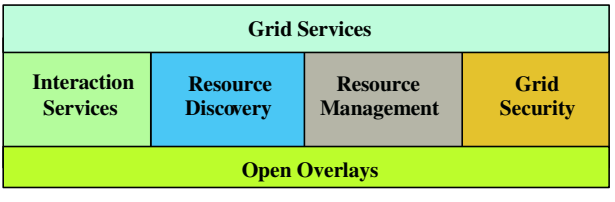


Fig. 1. The Gridkit Architecture

As illustrated in figure 1, Gridkit places a *Grid Services* layer, which is presented in terms of web services, on top of four orthogonal domains of generic middleware support. These, in turn, are layered on top of an *Open Overlays* layer which abstracts the diversity of underlying communications support mechanisms in a consistent manner. In more detail, the four middleware domains addressed by Gridkit are as follows:

1. *Interaction services*. This domain provides sophisticated and extensible application-layer communication services beyond SOAP: i.e., support for quality of service (QoS) configurable interactions, and for pluggable 'interaction types' such as publish-subscribe, multicast, streaming etc. More detail is available in [11].
2. *Resource discovery*. This offers generic and extensible resource and service discovery services. It supports the use of multiple discovery technologies to

maximise the flexibility available to applications. Examples of supported technologies are SLP or UPnP for traditional service discovery, Globus MDS for CPU discovery, and peer-to-peer protocols for general resource discovery.

3. *Resource management*. This domain, which is the focus of the present paper, provides comprehensive distributed resource management and scheduling support for Grid applications.
4. *Grid security*. This supports secure communication between participating nodes orthogonally to the interaction types in use.

The construction of Gridkit follows the OpenORB architectural approach [2] which is targeted at building dynamic and adaptive systems software. The general idea is as follows: a lightweight *component technology* called OpenCOM [6] provides building-blocks for constructing systems by composition (as advocated, e.g., by Szyperski [16]). *Reflection* then provides means to discover the structure and behaviour of component compositions, and to adapt and extend them at run-time. And, finally, *component frameworks* have the role of accepting 'plug-in' components, and ensuring architectural integrity during periods of adaptation. Each of the boxes in Fig. 1 is implemented in this way. As far as we know Gridkit is unique in applying such a component-based approach to both the middleware and the application layer of a Grid environment (most component-based systems, e.g. [9], address only applications).

3 The Resource Management Framework

3.1 Overview and Application Model

The distributed resource management design is realised as a component framework in which several areas of functionality are 'pluggable' as detailed below. At the most abstract level (see Fig. 2a), the framework is separated into two parts: *i*) a *global resource manager*, which coordinates resource management over multiple computational nodes, and *ii*) a *local resource manager*, which manages resources in an individual computational node, with two distinct phases: *i*) an initial *resource co-allocation* phase, and *ii*) a subsequent *run-time resource management* phase that can perform dynamic reconfiguration of resources in response to evolving application requirements and fluctuating resource availability in the infrastructure.

The framework requires that applications are structured and described as a graph of potentially-distributed OpenCOM components. In more detail, the description submitted to the framework by an application deployer (see Fig. 2b) is as follows[1]:

1. a set of top-level[2] OpenCOM components that encapsulate the various pieces of application functionality;
2. a set of QoS-annotated *tasks* (see below) which, among other things, express the required QoS of different areas of the application;

[1] This information is packaged as an XML schema which we do not present here due to space constraints.

[2] Components in OpenCOM may be *composite*—i.e. (recursively) composed of sub-components. In this paper, for simplicity, we only consider the special case of non-composite components. Essentially, composites are dealt with by applying the resource framework recursively.

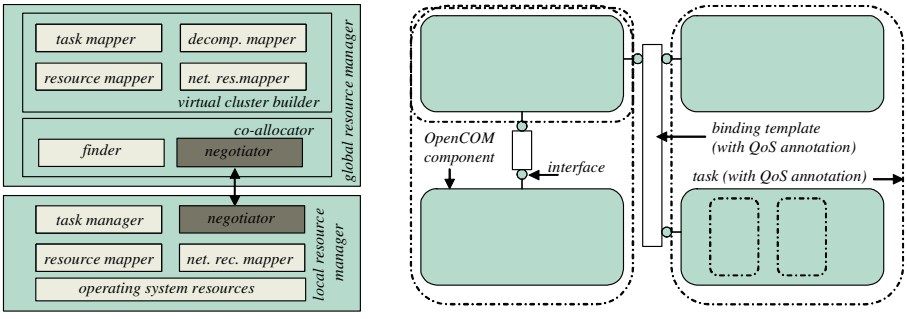


Fig. 2. a) Overall architecture. b) Elements of an application description

3. a set of QoS-annotated *binding templates* (see below) that represent bindings between the interfaces of the application components, and thus capture the abstract topology of the application as a graph;
4. a component/ task *mapping*—i.e. a list of components associated with each task.

Tasks are abstractions of 'activities' or 'units of work' which are meaningful at the application level and which can be decorated with application-oriented QoS annotations (the precise form of the QoS annotations is discussed below). Importantly, the definition of the tasks that comprise an application is *orthogonal* to the structure of the application itself in terms of components. Thus, in some cases (see Fig. 2b) a single task may span a set of (cooperating) components, while in others a single component may host multiple independent tasks (also, tasks may overlap, as shown). Examples: *i*) a 'transcode stream' task could be realised as a set of components (e.g. buffering, compressing, encoding etc.) that cooperate to transcode a media stream; *ii*) multiple instances of an 'access database' task could be encapsulated within a single component that deals with concurrent database access.

The orthogonality of tasks and component structure facilitates QoS specification that is meaningful at the application level. Thus, in the 'transcode stream' case, the QoS specification is attached to the entire user-visible task rather than to microcosmic aspects such as buffering etc. This orthogonality also offers a useful separation of concerns between writing an application and specifying its QoS.

Binding templates serve as abstract placeholders for inter-component communication bindings. A wide and extensible set of binding templates is supported including request-reply, multicast, publish-subscribe, workflow etc (see Fig. 2b). Binding templates are specified in terms of roles and message ordering constraints as described in detail in [15] and, like tasks, can be decorated with QoS annotations. Each binding template can be realised by (mapped to) a potentially wide range of concrete technologies (as supported by the interaction services module—see Fig. 1). For example, for components that will share a common node, a request-reply template could be realised as 'vtables' or as local IPC links; or, where the components will run on separate nodes, as TCP/IP connections or VME links. The precise mapping is selected by the decomposer as discussed in section 3.2.1.

Rather than define a fixed set of QoS parameters for use in QoS annotations, we support, for reasons of generality and extensibility, an extensible set of *QoS ontolo-*

gies that can be defined for specific areas of applicability. Each QoS ontology is a 'pluggable' entity that defines a vocabulary for QoS annotations, For example, a QoS ontology suitable for use with the 'transcode stream' task mentioned above might offer the following vocabulary: *frame_size, frame_rate: integer*. On the other hand, a simple QoS ontology for a request-reply binding template might offer: *connectivity: {high-speed, medium-speed, low-speed}*. As well as specifying a vocabulary, QoS ontologies are responsible for mapping QoS annotations to underlying pools of resources that are dedicated to individual tasks and bindings as discussed in the next section.

3.2 The Framework in Detail

The goal of the resource management framework is to place the application's constituent components on some appropriate set of physical computational nodes, and then to manage their ongoing execution. This involves the following steps: *i*) mapping components to configurations of virtual nodes called *virtual clusters*, *ii*) finding and negotiating the use of appropriate physical nodes and interconnecting with which to underpin virtual clusters, and *iii*) maintaining the QoS of the application at runtime in the face of fluctuating resource needs and resource provision.

3.2.1 Mapping Components to Virtual Clusters

Given an application description as defined in section 3.1, the *virtual cluster builder* (see Fig. 2a) is responsible for generating a list of candidate virtual clusters, each of which represents a possible physical infrastructure that could viably support the application. Virtual clusters could, for example, represent a set of processes on a machine, CPUs on a VME bus, islands of VME clusters, or machines randomly located in the global Internet.

The first step is that the *task mapper* derives the resource requirements of each of the application's tasks. To do this, it relies on the above-mentioned pluggable QoS ontologies which map QoS specifications expressed in their particular vocabularies to lower-level *resource ontologies*. As an example, the 'transcode stream' QoS ontology discussed in section 3.1 might map to a 'buffer processing' resource ontology with a vocabulary of *buffer_pool_size*, and *number_of_high_priority_threads*. Resource ontologies (which are hosted by the *resource mapper*) work analogously to QoS ontologies, but they be more general than a typical QoS ontology's so that they can underpin multiple QoS ontologies. In addition their vocabularies further map to the much lower-level expression: either concrete OS-level resources, or the vocabularies of further resource ontologies. So, extending the above example, the 'buffer processing' resource ontology might map to some quantity of memory, and a pool of threads with a given OS-level priority.

Following task and resource mapping, the *decomposition mapper* (with help from the network resource mapper—which is the network counterpart of the above-described resource mapper) generates a list of virtual cluster definitions that could potentially support the required resources according to various possible application decompositions. Rather than prescribe a fixed policy for decomposition, the decomposer accepts plug-in *decomposition heuristics* which are principally guided by three factors:

1. the resource requirements of each task (as derived above);
2. the set of available realisations of the binding templates specified in the application description;
3. the QoS annotations on the binding templates.

As an example, the goal of a particular decomposition heuristic might be to try to employ as few nodes as possible, and to depend on no more than low-speed connectivity between these nodes. This assumes the availability of well-resourced nodes that are each capable of supporting several tasks. The binding template information comes into play when it must be decided how to distribute the various tasks over multiple nodes. Likely 'lines of cleavage' can be identified where the components participating in one task are connected to components in an adjacent task only by a small number of bindings with 'relaxed' QoS requirements. For example, given the illustrative QoS ontology of section 3.1, a simple decomposition heuristic might try to map two adjacent tasks whose inter-task bindings were all "low speed", to two virtual nodes with a minimal interconnect (e.g. the global Internet). On the other hand, tasks whose inter-task bindings were "medium speed" or high speed" might need to be mapped to a common virtual node (or perhaps to two virtual nodes in a tightly-coupled cluster).

Overall, then, the output of the virtual cluster builder is a set of candidate virtual clusters. One of these candidate virtual clusters will eventually be selected and concretized to an appropriate infrastructure of physical clusters/ nodes/ interconnects etc. as explained below. When this concretization has been successfully performed, the rest of the candidate virtual clusters are discarded; however, the 'winning' virtual cluster is retained as a first-class entity throughout the subsequent runtime of the application as it plays a role in runtime adaptation (see section 3.2.3).

3.2.2 Finding and Negotiating Resources

Having derived a set of candidate virtual clusters, the *co-allocator* (see Fig. 2) is next invoked to locate appropriate sets of physical computational nodes with which to concretize candidate virtual clusters, and to negotiate the co-allocation of resources on these. The *finder*, which is part of Gridkit's resource discovery framework (see Fig. 1), is pluggable in terms of the mechanisms it uses for finding nodes: for example, a peer-to-peer search protocol could be used [14]; or alternatively, one could choose a simpler strategy such as querying a fixed set of available hosts for their current loading, or even querying a static central database of nodes à la Condor-G [8]. Having located a set of potentially suitable nodes, the co-allocator's *negotiator* confirms (or otherwise) the suitability of candidate physical nodes in supporting nodes from the virtual cluster. This is done by negotiating with each candidate's *local resource manager* (see Fig. 2a). Both end-system resources and the necessary network resources required by the concrete binding realization are considered in the negotiation.

The local resource management architecture is an extension of the work reported in [7]. In outline, the *task manager* is responsible for allocating resource pools to each of the application's tasks that will run on the machine, and for managing tasks at runtime (see below). The size and kind of these resource pools are determined by the resource mappers as discussed above. Separate instances of the task manager are created for each application running on the node.

3.2.3 Managing QoS at Runtime

At runtime, all application requests for resources (e.g. memory allocation, thread creation, or requests for abstract resources such as matrix containers that are understood by the framework) trap to the currently executing (runtime) task. Requests are then satisfied from the resource pools that were dedicated to the task when it was first instantiated.

QoS adaptation is triggered when the per-application task manager observes resource consumption or availability in some task falling outside acceptable thresholds that were determined during the QoS/ resource mapping phase. At this point, the task manager consults a plugged-in policy which specifies one or more of the following actions in some specified order:

- transfer resources from other tasks owned by the application;
- request more resources from the local operating system;
- negotiate with other task managers on the same machine (i.e. those associated with other applications) to borrow resources; or
- report to the parent virtual cluster, asking the latter to find new computational nodes on which to re-execute the affected task.

To support the latter action, the parent virtual cluster is also equipped with a plug-in policy capability to guide its subsequent actions.

A final avenue for QoS adaptation is an application-wide QoS renegotiation requested at the level of the global resource manager. In such cases, a revised application description is submitted and the mapping process is (selectively) restarted to attempt to meet the revised QoS requirements. In the absence of migratable components, this inevitably involves restarting certain application components. A deeper discussion of the fundamental research on which our runtime QoS management approach builds is available in [1].

3.3 Deployment Issues

The modular and configurable nature of the framework enables several deployment options in networked environments. One is to run the global resource manager on a small number of gateway machines, and to run the local resource manager on a much larger set of machines that are willing to contribute towards executing applications. Another option is to run both the global and local managers on some or all machines. Here, the global managers can interoperate in a peer-to-peer style in which an original application description is decomposed by global managers into sub-descriptions that can be delegated to other global managers (and so on recursively). This can result in faster deployment and also facilitate cooperation between peer virtual organisations. Beyond these two major options refinements are possible which leverage the underlying dynamic loading capabilities offered by OpenCOM [6]). For example, one can refine the first option to support just-in-time instantiation of the local manager on nodes as they are discovered by the co-allocator. Or, one can refine the second by dynamically instantiating the global functions on discovered nodes that know of distributed resources in their area but which do not wish to directly contribute themselves.

At a more fine-grained level, it is necessary to consider the deployment of the various plug-ins that are accepted by the framework. They would be the province of Grid

infrastructure experts with application-domain knowledge. Key plug-ins such as QoS and resource ontologies and decomposition heuristics would be expected to be produced relatively rarely, and be well documented and widely used by application developers, by this way to facilitate the *evolution* of the resource management framework.

4 Related Work

Apart from the Globus/ GRAM approach discussed in section 1, several researchers have attempted to alleviate the limitations identified in this paper. Condor-G [8] has been extensively used in the Globus context and provides a substantial instantiation of Globus/ GRAM. However, Condor-G supports only coarse-grained and concrete resource types, is statically configured and non-extensible, and has serious limitations in terms of adaptation: all it can do is migrate or restart jobs in the case of failures. ERDOS [4], has similar limitations in terms of resource types and configuration. In terms of adaptation it has a per-machine local 'resource' agent which monitors execution and reports exceptions to a global 'system manager'. However, this has the following drawbacks: *i*) adaptation is coarse-grained in that resource managers are per-machine rather than per-application (so that conflicting needs between applications are difficult to reconcile), and *ii*) local resource managers have no autonomy: all adaptation decisions are centrally made. Another feature of ERDOS is its notion of 'units of work'. These are superficially similar to our tasks; but in fact ERDOS units of work are equivalent to 'components': there is no orthogonality between the two concept as there is in our approach. GRMS [12] is a system with similar benefits and limitations to ERDOS. It is of interest in supporting on-the-fly querying of a fixed set of available hosts, thus obtaining more up to date information than Condor-G. However, there is no scope to configure other possible mechanisms such as peer-to-peer resource discovery.

More recently, [13] describes a resource management framework for 'interactive Grids'. This still suffers from resource type and lack of configurability, but it does have a more sophisticated local resource management scheme that features agents such as resource sensors for monitoring purposes and an enforcement agent for fulfilling QoS specifications. However, the work is limited in not supporting decomposition: each application can only run on a single computational node. A final piece of related work is research on the automatic decomposition of applications into workflows [5]. This takes an approach related to our decomposition mapper, but it lacks any support in the other areas addressed by our work or by the other systems discussed above.

In sum, the state of the art can fairly be characterised as lacking in *i*) fine-grained and abstract notions of QoS specification and resource provision, *ii*) plug-in configurability and extensibility, and *iii*) fine-grained runtime adaptation.

5 Conclusions and Future Work

We have discussed an approach to distributed resource management in the context of the Gridkit middleware platform. Our resource management framework promotes the

use of application-tailored abstract QoS specification which drives the allocation and adaptation of fine-grained and application-defined abstract resources. Furthermore, to better support individual applications and deployment environments, the framework is capable of being flexibly configured, extended and run-time reconfigured by means of plug-ins in the following areas: *i*) QoS ontologies, *ii*) resource ontologies, *iii*) application decomposition heuristics, *iv*) resource adaptation policies, and *v*) resource location strategies. In addition, it supports flexible and fine-grained runtime adaptation. And, finally, it supports the task concept which enables highly flexible mappings of applications onto distributed infrastructures, and also facilitates runtime adaptation. We believe that these features provide a foundation for a future QoS-driven and potentially autonomic resource management facility for the Grid.

Future work is planned on two fronts: first we will exercise and evaluate our framework by using it to support existing Grid applications. These include a distributed visualisation scenario from our partners at Oxford Brookes University, and a dynamic computationally-steered chemistry application from the RealityGrid project [3]. Second, we plan to explore autonomic self-management in Gridkit. This will build on the inherent openness of the (component-based) platform but will require additional frameworks that deal with areas such as monitoring, recovery strategy deployment, and recovery strategy selection. We have carried out initial explorations in this area and believe that Gridkit provides a highly promising context for these ideas.

References

1. Blair, L., Blair, G.S., Andersen, A., Coulson, G., Sanchez Ganedo, D., "Supporting Dynamic QoS Management Functions in a Reflective Middleware Platform", IEE Proceedings Software, Vol 147, No 1, pp. 13-21, 2000.
2. Blair, G.S., Coulson, G., Andersen, A., Blair, L., Clarke, M., Costa, F., Duran-Limon, H., Fitzpatrick, T., Johnston, L., Moreira, R., Parlavantzas, N., Saikoski, K., "The Design and Implementation of OpenORB v2", IEEE DS Online, Special Issue on Reflective Middleware, Vol. 2, No. 6, 2001.
3. Brooke, J.M., Coveney, P.V., Harting, J., Jha, S., Pickles, S.M., Pinning R.L., Porter, A.R., "Computational Steering in RealityGrid", Proc. UK e-Science All Hands Meeting, 2003, http://www.nesc.ac.uk/events/ahm2003/AHMCD.
4. Chatterjee, S., Sabata, B., Brown, M., "Adaptive QoS Support for Distributed, Java-based Application" In Proceedings of the IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORK), St-Malo, France,1999.
5. Cicerre, F., Madeira, E., Buzato, L., "A Hierarchical Process Execution Support for Grid Computing", 2nd Intl Workshop on Middleware for Grid Computing, Toronto, Canada, October 2004.
6. Clark, M., Blair, G.S., Coulson, G., Parlavantzas, N., "An Efficient Component Model for the Construction of Adaptive Middleware", Proc. IFIP Middleware 2001, Heidelberg, Germany, November 2001.
7. Duran-Limon, H., Blair G.S., "Reconfiguration of Resources in Middleware", 7th IEEE International Symposium on Object-oriented Real-time Dependable Systems (WORDS 2002), San Diego, CA, January 2002
8. Frey, J., Tanenbaum, T., Livny, M., Foster, I., Tuecke,S., "Condor-G: A Computation Management Agent for Multi-Instructional Grids", Cluster Computing, Vol 5, pp237-246, 2001.

9. Furmento, N., Mayer, A., McGough, S., Newhouse, S., Field, T., Darlington, J., "ICENI: Optimisation of Component Applications within a Grid Environment", Parallel Computing, Vol 28, No 12, pp1753-1772, 02.
10. The Globus Project, "Resource Management: The Globus Perspective", presentation at GlobusWorld 2003, available at http://www.globus.org/, 2003.
11. Grace, P., Coulson, G., Blair, G., Mathy, L., Yeung, W.K., Cai, W, Duce, D., Cooper, C., "GRIDKIT: Pluggable Overlay Networks for Grid Computing", to appear in Proc. Distributed Objects and Applications (DOA'04), June 2004.
12. Huang, J., Wang, Y., Cao, F. "On Developing Distributed Middleware Service for QoS- and Criticality-Based Resource Negotiation and Adaptation", Special issue on Operating Systems and Services, Journal of Real-Time Systems,1998.
13. Kumar, R., Talwar, V., Basu, S., "A Resource Management Framework For Interactive Grids", 1st Intl Workshop on Middleware for Grid Computing, Rio de Janeiro, Brazil, June 2003.
14. Pallickara, S., Fox, G., "NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids", Proc. IFIP/ACM/USENIX Middleware 03, Rio de Janeiro, Brazil, April 2003.
15. Parlavantzas, N., Coulson, G., Blair, G.S., "An Extensible Binding Framework for Component-Based Middleware", Proc. Enterprise Distributed Object Computing Conference (EDOC 2003), Brisbane, Australia, Sept. 2003.
16. Szyperski, C., "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, 1998.

Stochastic Approach for Secondary Storage Data Access Cost Estimation

Lukasz Dutka[1] and Jacek Kitowski[1,2]

[1] Academic Computer Center CYFRONET AGH, Cracow, Poland
[2] Institute of Computer Science, AGH-UST, al.Mickiewicza 30, 30-059, Cracow, Poland
phone: (+48 12) 6173964, *fax:* (+48 12) 6338054
{dutka, kito}@agh.edu.pl

Abstract. This paper is a summary of experimental results in the domain of cost estimations of data access to data stored in the secondary storage. It mainly discusses technical and development issues based on implementation prepared for EU CrossGrid project. The proposed solution is an extension of research in the field of optimization of data access for grid environment, published previously elsewhere. It is important to mention, that the presented solution might seem very simple in comparison with already available ones based on the analytical secondary storage simulation approach, but in contrast to them this method can give precise predictions even if the information about the secondary storage internal architecture is uncertain and incomplete.

1 Introduction

Certain scientific applications from domains such as High-Energy Physics, Flood Crisis Team Support, Biomedical Research, developed within EU funded projects, like Crossgrid and European Data Grid [1, 2] are expected to produce Tera- or even Petabytes of data that are analyzed and evaluated by scientists all over the globe. In order to achieve high level of availability and fault tolerance, as well as minimal access time for these large data volumes, data replication is applied very frequently. Several grid projects have implemented data replication systems that handle these requirements partially. The task of a replica management system is not only to keep track of the replicas but also to select those replicas that can be accessed by an application with a minimal response or transfer time. In typical data grids large amounts of replicated data are stored in different storage systems with access latencies ranging from seconds to hours; the latter relates to data residing on a tape that is not mounted yet. Thus, it is necessary to provide specific tools to predict the access time of data intensive jobs in such heterogeneous environments.

This paper focuses on cost estimation of access to the secondary storage (e.g., constructed of hard drives including disk arrays, SAN drives, etc.). It extends our previous studies on usage of the tertiary storage in grids [3, 4]. So, in the reported work we do not deal with long access times but, instead, with the heterogeneity

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 796–804, 2005.

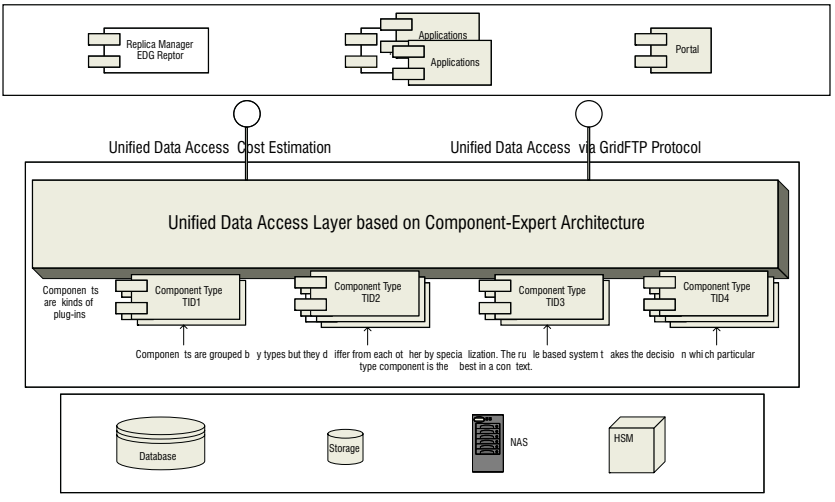


Fig. 1. Overview of Unified Data Access Layer approach

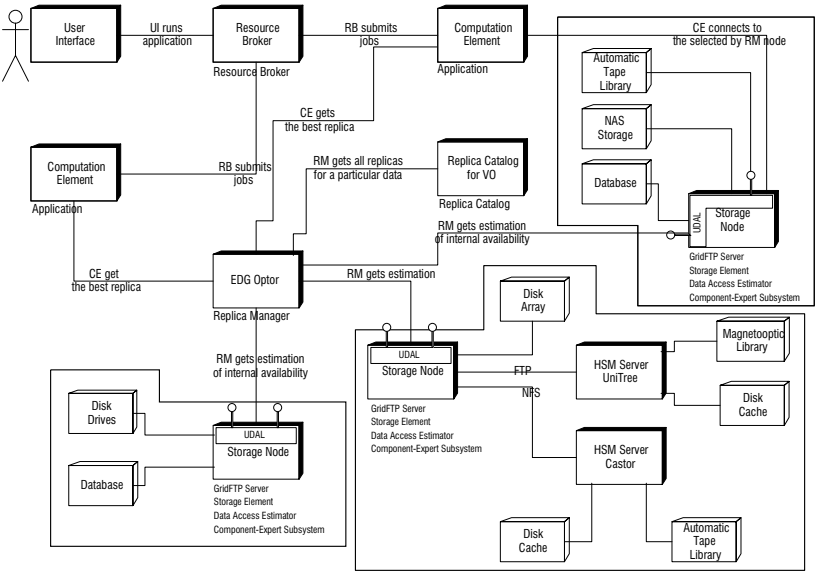


Fig. 2. Example of UDAL deployment in EU CrossGrid Project [9]

of storage devices and different efficiency of machines controlling access to those devices, that make the estimation process difficult.

The estimator for the secondary storage described in this paper is a component for Unified Data Access Layer (UDAL) (see Fig. 1) compatible with Component-Expert Architecture [5, 6, 7, 8]. It simply can be understood as a

plug-in to UDAL automatically selected by semi-intelligent infrastructure when necessary. See example of UDAL usage for the CrossGrid Project [9] in Fig. 2.

The paper is organized as follows. In Section 2 we provide an overview of our approach to data access cost estimation for the secondary storage. Experimental results are shown in Section 3, we conclude the paper and provide some insight into future work in Section 4.

2 Data Access Cost Estimation for Secondary Storage

Due to heterogeneity of storage devices used for construction of the secondary storage as well as due to complexity of software controlling those devices, it is difficult, and in some cases even not possible, to design estimator based on simulation approach similar to that one for tertiary storages [3, 10, 11]. It assumes that we are able to simulate behavior of hardware and software. Taking into account the secondary storage based on Storage Area Network (SAN) devices, it is not possible to predict future load since requests can come from the entire SAN network; moreover, the grey-box approach [3, 10, 11] used for simulation of the tertiary storage is usually limited in use for SAN. Additionally, since, the Grid is considered as a very dynamic environment with highly fluctuating fabric infrastructure and distributed management, a simple method is required for predicting access cost of different kinds of grid storage in order to make a proper selection of replicas, which are going to be used as input data for calculations. Analytical models in such a fuzzy environment are hard to be used due to the dynamics and uncertain states. Most of the analytical models are based on detailed disk drive simulators widely discussed by Ruemmler & Wilkes [12], Kotz [13] or Ganger [14]. Another popular approach, maybe a bit closer to the one we present within this paper, is extracting disk parameters by microbenchmarking presented by Talagala [15] or Worthington [16]. However, in both cases these approaches cannot be involved in data access costs prediction for production systems since their algorithms are too complex to be involved for estimations on-demand as well as the state of data allocation is not taken into account.

Active experiments, like probing (contrary to the passive ones and to the static analytical models [17, 18, 19] – often mentioned in state of the art), that are based on a typical approach taken from the control theory (identification of the models) seem to be more appropriate, since they allow the user to get the performance information in a current state for a given data size. Moreover, such an approach can take into account current loads and the available bandwidth in the grid which could be useful for automatic replica selection or creation. Another important feature is to consider disk subsystems, for which the set of parameters needed for the analytical description of the performance have not been defined yet, as well as the subsystems with partial failure (e.g. with number of bad tracks increasing or need for frequent heads repositioning).

Taking into account those issues, we decided to start research on the approach which we call 'probing', contrary to simulation used for the tertiary storage. Obviously, in most real cases it is not possible to probe access cost to all blocks

Table 1. Number of required probing retries to predict access cost

	Min. number retries	Max. number retries
less then 1MB	1000	2000
1MB - 100MB	50	100
more then 100MB	40	70

which files are consisted of, due to tight limitation for the whole time of operation execution. We assumed that the estimation time should be within 0.5s range. So probing must be done for selected number of disk blocks only. However, it has been verified, that simple reading of first x number of blocks is inappropriate – unpredictable buffering and caching algorithms realized by operating systems as well as by devices make estimations obtained in this way useless.

Therefore, we introduce a statistical approach based on random selection of representation of blocks of data to probe, in order to obtain a representation of the access time to the entire data. This approach can produce accurate estimations useful for the short-term usage. From the practical point of view, in the secondary storage case, the most important factor, which we want to predict, is the data transfer bandwidth from a disk device to memory of the machine controlling this particular device. Contrary to the tertiary storage, the latency is not as important since, the secondary storage usually can provide data in less then 10ms. Paradoxically, the case of small and very small files is more difficult to assess, since probing of access cost can usually lead to buffering data by the operating system, that influences significantly the estimation results. However, the system buffering decision is hard to predict. The situation is even more complicated, when we deal with more intelligent disk arrays which can offer some read-ahead and other features for efficiency improving. To solve that problem we decided to bind parameters of the stochastic experiment with the file size. Basing on deep analysis of Linux file buffering approach and some more popular disk arrays providing I2O functionality we constructed a table of minimum and maximum retries (limitations of probe events), selected heuristically, required to perform on the file in order to get most accurate estimation (see Table 1).

Our estimation algorithm looks as follows:

```
// select randomly number of retries between
// min and and max from table
retry=minRetries+(maxRetries-minRetries)*rand()/(RAND_MAX+1.));

// size of sample block should be less then 1MB
if (currentFileSize<1024*1024){
    maxBlockSize = currentFileSize;
}else{
    maxBlockSize = 1024*1024;
}
```

```
ftime(&t_start); // start stopper

for (int x=0;x<retry;x++){
    // select randomly size of the probed block.
    // Value between 1 and maxBlockSize
    sampleSize=1+maxBlockSize*rand()/(RAND_MAX+1.0));
    if (sampleSize >currentFileSize){
        sampleSize = currentFileSize;
    }

    // select randomly start position of probed block
    sampleOffset=((currentFileSize-sampleSize)*rand()
                  /(RAND_MAX+1.0));

    lseek(fd, sampleOffset, SEEK_SET);
    lBytesRead+=read(fd, buff, sampleSize);
}//for

ftime(&t_current); // stop stopper

//total time of reading lBytesRead bytes is t_diff
t_diff = computeTimeDiff(t_start, t_current);
```

To summarize the algorithm above, does the following steps:

- Select randomly number of probing retries.
- Define maximal size of probing blocks for the experiment, which obviously must be lower then the entire file size.
- Starting stopwatch.
- In the loop repeating the following actions:
  - Select randomly size of the probed block. It should be between 1 byte and previously determined maximal size.
  - Select randomly start position for reading of data. Simply, it must be between beginning of the file end minus current block size we are going to read.
  - Reading of data and storing number of read bytes in a global variable.
- Stopping stopwatch.
- Computation of time spent for the experiment, what can be directly used for calculation of the most important factor - data bandwidth.

3 Experimental Results

In the following section we provide experimental results of the data access cost estimation using the estimator described in Section 2 and deployed on two different machines `zeus04` and `zeus25` located in CYFRONET-AGH (Poland) [20],

where zeus04 was fully under our control and zeus25 was a production Storage Element for CrossGrid infrastructure, being used simultaneously by different users during the experiment.

The zeus04 machine was equipped with one single hard disk drive and zeus25 had a hard disk drive and a disk array connected to it using NFS. The disk array was also shared by other machines in zeus cluster, which was composed of 30 nodes. Both machines provided access to their files through GridFTP server. For real cost measurements execution time of the following command

```
zeusXX~\$~ globus-url-copy gsiftp://zeusXX//FileName.dat
           file:///dev/null
```

was adopted.

The command above was always executed on the same machine from which data was transferred outside, since the estimator estimates only the access cost inside storage elements (for global access cost estimation issues see [3, 4]).

The cost estimation experiments have been performed on a set of files, f1KB.dat, f1MB.dat, f100MB.dat, f1GB.dat with sizes: 1KB, 1MB, 100MB and 1GB, respectively. That set of files was replicated into the zeus04 and zeus25 hard disk drives as well as into the SAN device connected to zeus25 via NFS protocol.

Using that hardware configuration we performed an experiment on verification of accuracy of the estimations. Since, the experiment was made on the production hardware environment, it was necessary to perform each step several times. Thus, we executed every single test 100 times for calculation of the average errors.

Tables 2, 3 and 4 present gathered results for bandwidth estimations for different devices.

We can see that the absolute error for the experiments is less then 10% considered as a very good result. Comparison of results for HDD and NFS shows higher accuracy obtained for the latter case. This feature is probably caused by lower involvement of the operating system in data buffering/caching in this case in comparison with the case considering pure HDD drive.

Another important conclusions is that the accuracy of estimations depends on the size of files. Obviously, the bigger files got less accurate estimations, due to buffering performed by the operating systems. Some blocks of the processed file are buffered, while some are not, which results in faster or slower access then estimated using only on some random subset of blocks. Additional reason

Table 2. Experimental results for files located on zeus25 HDD drive

	Average Estimated Bandwidth	Average Real Bandwidth	Absolute Error
f1KB.dat	356.20 MB/s	380.23 MB/s	6.32 %
f1MB.dat	164.76 MB/s	159.11 MB/s	3.55 %
f100MB.dat	19.24 MB/s	20.29 MB/s	5.17 %
f1GB.dat	19.00 MB/s	21.02 MB/s	9.61 %

Table 3. Experimental results for files located on zeus25 NFS SAN drive

	Average Estimated Bandwidth	Average Real Bandwidth	Absolute Error
f1KB.dat	281.12 MB/s	278.62 MB/s	0.90 %
f1MB.dat	123.57 MB/s	128.15 MB/s	3.57 %
f100MB.dat	15.66 MB/s	14.96 MB/s	4.67 %
f1GB.dat	12.92 MB/s	12.64 MB/s	2.22 %

Table 4. Experimental results for files located on zeus04 HDD drive

	Average Estimated Bandwidth	Average Real Bandwidth	Absolute Error
f1KB.dat	159.46 MB/s	161.39 MB/s	1.19 %
f1MB.dat	136.37 MB/s	147.09 MB/s	7.29 %
f100MB.dat	24.15 MB/s	22.97 MB/s	5.13 %
f1GB.dat	23.41 MB/s	21.34 MB/s	9.68 %

for results discrepancy is file fragmentation, more evident for bigger files. It is important to remark that the huge transfers obtained for the smallest files described in the tables above, are thanks to the operating system cache. Moreover we can see that even though, the prediction are very precise.

4 Conclusions

In this paper we thoroughly described our approach for estimation of data access cost for data stored in the secondary storage.

Active experiments, like probing, that are based on a typical approach taken from the control theory (identification of the models) seem to be more appropriate to get the performance information in a current state for a given data size. Additional advantages of the approach are: consideration of current loads and the available bandwidth in the grid (useful for automatic replica selection or creation), assessment of grid storage subsystems for which the parameters are unavailable or uncertain as well as the subsystems with partial failure, difficult to follow analytically. For the long-term development of the approach one can also see the possibility of implementation of experience management/learning approach in order to extend precision of the subsystem performance prediction in a real grid environment. Even if data size is small making use of the disk cache, this fact is taken into account by appropriate organization of probing.

Summing up, in our opinion the accurate prediction of the access cost to the secondary storage achieved in our experiments shows that even so simple to implement method can be very effective in practice, and the analytical models should be used for environments where the probing approach is not feasible like the tertiary storage. A kind of proof of concept, presented in this paper, requires more work and testing performed in production environment, which is planned to be completed on grid testbeds available.

Acknowledgements

The work described in this paper was supported in part by the European Union through the IST-2001-32243 project "CrossGrid". AGH grant is also acknowledged.

References

1. Crossgrid - development of grid environment for interactive applications, 2001. EU Project no.: IST-2001-32243, http://www.crossgrid.org.
2. The European DataGrid Project. `http://www.edg.org`.
3. L. Dutka, R. Slota, D. Nikolow, and J. Kitowski. Optimization of Data Access for Grid Environment. In *1st European Across Grids Conference*, Lecture Notes in Computer Science, no. 2970, pages 93–102, Universidad de Santiago de Compostela, Spain, February, 13-14 2003. Springer.
4. K. Stockinger, H. Stockinger, L. Dutka, R. Slota, D. Nikolow, J. Kitowski. Access Cost Estimation for Unified Grid Storage Systems. In *4th International Workshop on Grid Computing (Grid2003)*, Phoenix, Arizona, November 17 2003. IEEE Computer Society Press.
5. L. Dutka and J. Kitowski. Application of Component-Expert Technology for Selection of Data-Handlers in CrossGrid. In D. Kranzlmüller, P. Kacsuk, J. Dongarra, and J. Volkert, editors, *Proc. 9th European PVM/MPI Users' Group Meeting*, volume 2474 of *Lecture Notes on Computer Science*, pages 25–32. Springer, Sept. 29 - Oct. 2 2002.
6. L. Dutka and J. Kitowski. Flexible Component Architecture for Information WEB Portals. In A. Bogdanov J. Dongarra A. Zomaya Y. Gorbachev P. Sloot, D. Abramson, editor, *Proc. Computational Science - ICCS 2003 International Conference*, Lecture Notes in Computer Science, no. 2657, pages 629–638, Saint Petersburg Russian Federation, Melbourne Australia, June 2 - 4 2003.
7. L. Dutka and J. Kitowski. Expert Technology in Information Systems Development Using Component Methodology. In *Proc. of Methods and Computer Systems in Science and Engng.*, pages 199–204, Cracow, Nov.19-21 2001. ONT. (in Polish).
8. L. Dutka, R. Slota, and J. Kitowski. Component-Expert Architecture as Flexible Environment for Selection of Data-handlers and Data-Access-Estimators in CrossGrid. In M. Turala M. Bubak, M. Noga, editor, *Proceedings Cracow Grid Workshop '02*, pages 201–209, Cracow, Poland, December 11-14 2002 2002.
9. The CrossGrid Project. `http://www.crossgrid.org`.
10. D. Nikolow, R. Slota, M. Dziewierz, and J. Kitowski. Access Time Estimation for Tertiary Storage Systems. In R. Feldman B. Monien, editor, *Euro-Par 2002 Parallel Processing, 8th International Euro-Par Conference*, volume 2400 of *Lecture Notes on Computer Science*, pages 873–880, Paderborn, Germany, August 27-30 2002. Springer.
11. D. Nikolow, R. Słota, and J. Kitowski. Data access time estimation for hsm systems in grid environment. In *Proceedings of the Cracow Grid Workshop*, December 2002.
12. Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–28, March 1994.
13. David Kotz, Song Bac Toh, and Sriram Radhakrishnan. A detailed simulation model of the HP 97560 disk drive. Technical report, 1994.

14. Gregory R. Ganger and John S. Bucy. The disksim simulation environment, January 24 2003.
15. N. Talagala, R. Arpaci-Dusseau, and D. Patterson. Microbenchmark-based exctraction of local and global disk characteristics. web, 1999. http://www.cs.wisc.edu/ remzi/Postscript/disk.pdf.
16. B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes. On-line extraction of SCSI disk drive parameters. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 146–156, Ottawa, Canada, May 15–19 1995.
17. J.S. Bucy and G.R. Ganger. *The DiskSim Simulation Environment Version 3.0 Reference Manual*. School of Computer Science Carnegie Mellon University, Pittsburgh, 2003.
18. J. Schindler and G.R. Ganger. Automated Disk Drive Characterization. Technical report, School of Computer Science Carnegie Mellon University, Pittsburgh, December 1999. CMU SCS Technical Report CMU-CS-99-176.
19. E. Bachmat J. Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *Proceedings of SIGMETRICS 2002 Conference*, Marina Del Rey, California, June 2002.
20. Academic Computer Center - CYFRNET AGH, Poland. `http://www.cyfronet.krakow.pl`.

A Cluster-Based Dynamic Load Balancing Middleware Protocol for Grids

Kayhan Erciyes[1,2] and Reşat Ümit Payli[3]

[1] Izmir Institute of Technology, Computer Eng. Dept.,
Urla, Izmir 35430, Turkey
[2] California State University San Marcos,
Computer Science Dept. San Marcos CA 92096, U.S.A
kerciyes@csusm.edu
[3] Computational Fluid Dynamics Laboratory,
Purdue School of Engineering and Technology,
Indiana University-Purdue University,
Indianapolis, Indiana 46202, U.S.A
rpayli@iupui.edu

Abstract. We describe a hierarchical dynamic load balancing protocol for Grids. The Grid consists of clusters and each cluster is represented by a coordinator. Each coordinator first attempts to balance the load in its cluster and if this fails, communicates with the other coordinators to perform transfer or reception of load. This process is repetaed periodically. We show the implementation and analyze the performance and scalability of the proposed protocol.

1 Introduction

Computational Grids consist of heterogenous computational resources, possibly with different users, and provide them with remote access to these resources [1], [2], [3]. The Grid has attracted reserachers as an alternative to supercomputers for high performance computing. One important advantage of Grid computing is the provision of resources to the users that are locally unavailable. Since there are multitude of resources in a Grid environment, convenient utilization of resources in a Grid provides improved overall system performance and decreased turnaround times for user jobs [4]. Users of the Grid submit jobs at random times. In such a system, some computers are heavily loaded while others have available processing capacity. The goal of a load balancing protocol is to transfer the load from heavily loaded machines to idle computers, hence balance the load at the computers and increase the overall system performance. Contemporary load balancing algorithms across multiple/distributed processor environments target the efficient utilization of a single resource and even for algorithms targetted towards multiple resource usage, achieving scalability may turn out to be difficult to overcome.

A major drawback in the search for load balancing algorithms across a Grid is the lack of scalability and the need to acquire system-wide knowledge by

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 805–812, 2005.

the nodes of such a system to perform load balancing decisions. Scalability is an important requirement for Grids like NASA's Information Power Grid (IPG) [5]. Some algorithms have a central approach [6], yet others require acquisition of global system knowledge. Scheduling over a wide area network requires *transfer* and *location* policies. Transfer policies decide *when* to do the transfer [7] and this is typically based on some threshold value for the load. The location policy [8] decides *where* to send the load based on the system wide information. Location policies can be *sender initiated* [9] where heavily loaded nodes search for lightly loaded nodes, *receiver initiated* [10] in which case, lightly-loaded nodes search for senders or *symmetrical* where both senders and receivers search for partners [11]. Load balancing across a Grid usually involves sharing of data as in an MPI (Message Passing Interface) *scatter* operation as in [12], [13]. MPICH-G2, is the a Grid-enabled implementation of MPI that allows a user to run MPI programs across multiple computers, at the same or different sites, using the same commands that would be used on a parallel computer [14].

We propose a protocol to perform load balancing in Grids dynamically where an ordinary node does not need to have a global system wide knowledge about the states of other nodes in the Grid. The protocol is semi-distributed due to the existence of local cluster center nodes called the *coordinators*. We show that the protocol designed is scalable and distributed as the coordinators communicate and synchronize asynchronously.

The paper is organized as follows: In Section 2, the proposed protocol including the coordinator and the node algorithms is described with the analysis. In Section 3, the implementation of the protocol using an example is explained and Section 4 contains the concluding remarks along with discussions.

2 The Protocol

For load balancing in grids, we propose the architecture shown in Fig. 1 where nodes form clusters and each cluster is represented by a coordinator similar to [15]. Coordinators are the interface points for the nodes to the ring and perform load transfer decisions on behalf of the nodes in their clusters they represent. They check whether load can be balanced locally and if this is not possible, they search for potential receivers across the Grid. *Load* can be specified in many different ways. One common approach is the count of the processes waiting in the *ready queue* of the processor. The only resource required by a ready process to execute is the processor. When the count value is detected to be higher than the *upper threshold*, we say that the node is HIGH, otherwise when the number of processes in the ready queue is below a *lower threshold*, the node is LOW meaning it can accept load from the other nodes. A ready queue at a node can have a value in between these two thresholds in which case the node is considered MEDIUM. We also assume that only *non-preemptive* transfers are possible which would indicate that only processes that have not started execution in the host node can be transferred along with their data. For the protocol, we will concentrate on the mechanism to decide *when* and *where* the transfer of load

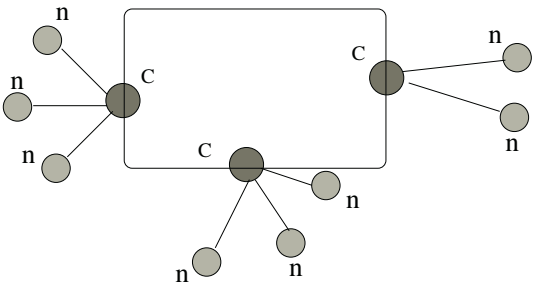


Fig. 1. The Load Balancing Model for a Grid

should be performed rather than *how*. It should also be noted that the protocol will work equally for sharing of data across the Grid for parallel applications as in scattering operations.

2.1 Coordinator Algorithm

The coordinator is responsible to monitor local loads, initiate transfer from HIGH to LOW nodes if there are local matches and search for LOW nodes across the Grid if there are no local matches. Its state diagram is depicted in Fig. 2.

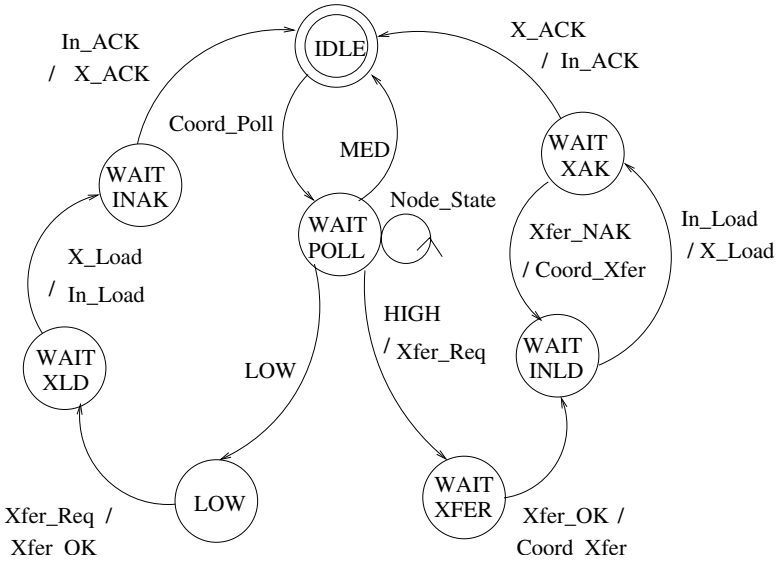


Fig. 2. Coordinator Algorithm State Machine

```
Process Coordinator;
1.  Begin
2.   While TRUE
3.     Wait for Time_Out;
4.     Send Coord_Poll messages to nodes,
5.     Receive Loads from nodes;
6.     If there are local matches
7.        Send Coord_Xfer to HIGH nodes;
8.     Else
9.       Send Xfer_Request message to next Coordinator;
10.       If Xfer_OK received Send Coord_Xfer to HIGH node;
11.       If Load is received Xfer Load to next Coordinator;
12.       If Xfer_ACK Send Xfer_ACK to HIGH node;
13.         Else Send Coord_Xfer to HIGH Node;
14. End.
```

Fig. 3. Coordinator Algorithm Pesudocode

It is awaken by a timer interrupt and sends a $Coord\_Poll$ message to every node in its cluster. When it receives the loads, it checks whether there are any local matching nodes. If there is a match, it sends $Coord\_Xfer$ message to HIGH node to initiate transfer. Otherwise, it sends a $Xfer\_Request$ message to the next coordinator in the ring and waits for a reply. If it receives the original message back, there are no matches and the next period is waited. If there is a remote match ($Xfer\_OK$), the coordinator initiates transfer from the local node by sending $Coord\_Xfer$. When it receives the load from the local node, it transfers this to the target coordinator which then passes the load to the target node. If transfer is error free, the target node responds by $Xfer\_ACK$ which is passed along the coordinators to the source node. The pseudocode for the coordinator algorithm is depicted in Fig.3.

2.2 Node Algorithm

The node process sends its load to the coordinator when it receives the $Coord\_Poll$ message from the coordinator. If its load is HIGH, it waits for initiation of transfer from the coordinator. When it receives this initiation ($Coord\_Xfer$), it sends the excessive load to the specified receiver and then waits for acknowledgement. If there is an error in transfer, the process is repeated as shown in Fig.4. If its load is LOW, it will wait until a transfer from the HIGH node or a timeout.

2.3 Analysis

Let us assume k, m, n and d are upperbounds on the number of clusters, nodes in a cluster in the network, nodes in the ring of coordinators and the diameter of a cluster respectively.

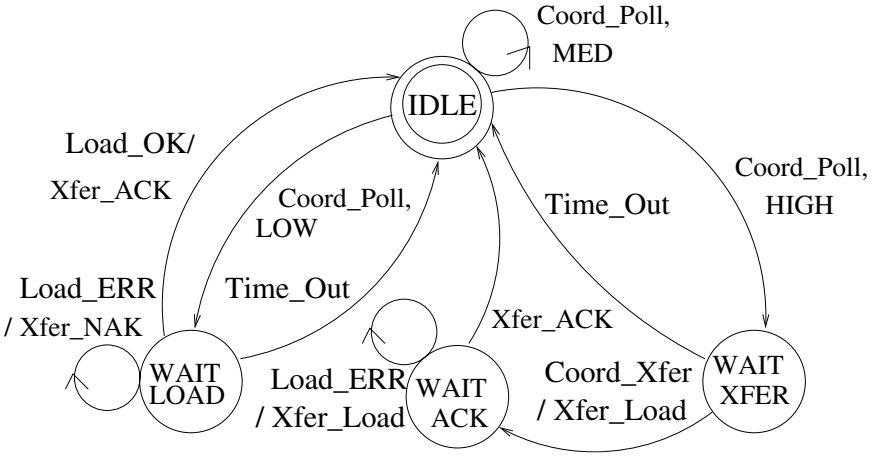


Fig. 4. Node Algorithm State Machine

Theorem 1. *The total time for a load transfer is between* $4dT + L$ *and* $(4 + k)dT + L$ *where* T *is the average message transfer time between adjacent nodes and* L *is the actual average load transfer time.*

Proof. A node transfers its state to the coordinator in d steps in parallel with the other nodes and assuming there is a match of LOW-HIGH nodes in the local cluster, the coordinator will send $Coord\_Xfer$ message to the HIGH node in d steps. Then there will be L time for the actual load transfer. The HIGH and LOW nodes also perform a final handshake to confirm delivery of load in $2d$ steps. The total minimum time for load transfer is then the sum of all of these steps which is $4dT + L$. In the case of a remote receiver, the messages for load transfer have to pass through k hops resulting in $(4 + k)dT + L$ time.

Corollary 1. *The total number of messages exchanged for load transfer is* $O(k)$.

Proof. As shown by Theorem 1, the maximum total number of messages required for a remote receiver will be $(4 + k)d$. Assuming d is approximately unity, the message complexity of the algorithm is $O(k)$.

Corollary 2. *The protocol described achieves an order of magnitude reduction in the number of messages exchanged for load transfer with respect to a similar protocol that does not use clusters.*

Proof. Assuming $k{=}m$, that is, the maximum number of clusters in the Grid equals the maximum number of nodes in a cluster, the total number of messages

exchanged would be in the order of $O(k^2)$ for a similar protocol that does not use any hierarchical cluster structure. Therefore, the number of messages using our approach provides an order of magniude decrease in the number of messages transferred.

3 An Example Operation

An example operation of the model is depicted in Fig.5. The following are the sequence of events :

1. All of the nodes in clusters 1, 2 and 3 inform their load states to their cluster coordinators C_1, C_2 and C_3. There are no LOW nodes in Cluster 1, there is one HIGH and one LOW node in Cluster 2 and 1 LOW and two MED nodes in Cluster 3. This is shown in Fig.5(a).
2. The coordinator for Cluster 1, C_1, forms a request message $Xfer\_Req$ and sends it to the next coordinator on the ring, C_2.
3. C_2 has a local match between its two nodes (n_{23} and n_{21}) and has no other receivers. It therefore passes the message immediately to its successor C_3. It also sends $Coord\_Xfer$ message to n_{21} to initiate local load transfer.
4. C_3 has a potential receiver (n_{32}) which has reported LOW load. It therefore replies by changing the message $Xfer\_Req$ to $Xfer\_OK$ and sends this to C_1. Steps 2,3 and 4 are shown in Fig.5(b).

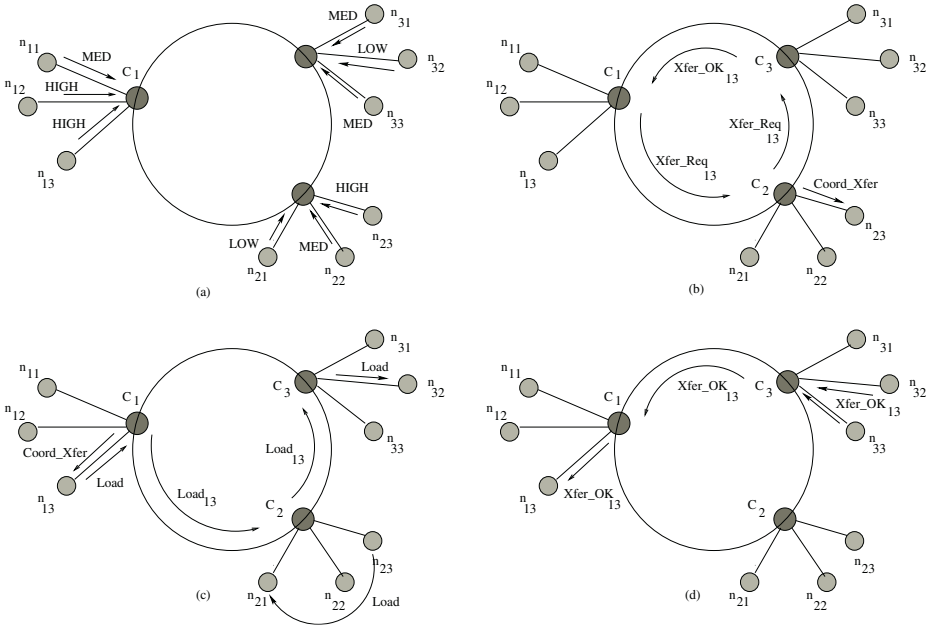


Fig. 5. An Example Operation of the Protocol

5. C_1 receives the reply message for R_{13} and sends the *Coord_Xfer* message to n_{13} which then sends its load to C_1.
6. C_1 now transfers the load to C_3 which transfers the load to n_{32}. Steps 5 and 6 are shown in Fig.5(c).
7. n_{32} sends $Xfer\_ACK$ message to its coordinator C_3 which passes this to C_1 which then forwards it to n_{13}. This step is shown in Fig.5(d).

4 Discussions and Conclusions

We proposed a framework and a protocol to perform dynamic load balancing in Grids. The Grid is partitioned into a number of clusters and each cluster first tries to balance the load locally and if this is not possible, a search for potential receivers is performed across the Grid using a sender-initiated method. We showed that the proposed protocol is scalable and has significant gains in the number of messages and the time perform load transfer. We have not addressed the problem of *how* the load should be transferred but we have tried to propose a protocol that is primarily concerned on *when* and *where* the load should be transferred. In fact, it may be possible just to transfer data part of the load by employing copies of a subset of processes across the nodes in the Grid. Load balancing across a Grid, in a general sense, would involve transferring of data for parallel applications.

The coordinators have an important role and they may fail. New coordinators may be elected and any failed node member can be excluded from the cluster. The recovery procedures can be implemented using algorithms as in [16] which is not discussed here. Our work is ongoing and we are looking into implementing the proposed structure in a Grid with various load simulations. Another research direction would be the investigation of the proposed model for real-time load balancing across the Grid where load balancing decisions on where and when to do the transfer and the actual load transfer should be performed before a pre-determined soft or hard deadlines. Yet another area of concern is keeping the copies of a subset of processes at nodes (*shadow processes*) to ease load transfer.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. Journal of High Performance Computing Applications, 15(3), (2001), 200-222.
2. Foster, I.: What is the Grid ? A Three Point Checklist, Grid Today, 1(6), (2002).
3. Foster, I., Kesselman, C., eds.: The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Fransisco, CA 1999.
4. Arora, M., Das, S., K., Biswas, R.: A De-centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments, Proc. of Int. Conf. Parallel Processing Workshops, (2002), 499-505.

5. Johnston, W. E., Gannon, D., Nitzberg, B.: Grids as Production Computing Environments : The Engineering Aspects of NASA's Information Power Grid. Proc. 8th Int. Sym. High Performance Distributed Computing, (1999), 197-204.
6. Akay, O., Erciyes, K.: A Dynamic Load Balancing Model For a Distributed System. Journal of Mathematical and Computational Applications ,Vol.8, 2003, No:1-3.
7. Eager, D. L., Lazowska, E. D., Zahorjan, J.: A Comparison of Receiver-initiated and Sender-initiated Adaptive Load Sharing, Performance Evaluation, 6(1), (1986), 53-68.
8. Kumar, V. , Garma, A., Rao, V.: Scalable Load Balancing Techniques for Parallel Computers, Journal of Parallel and Distributed Computing, 22(1), (1994), 60-79.
9. Liu, J., Saletore, V. A.: Self-scheduling on Distributed Memory Machines, Proc. of Supercomputing, (1993), 814-823.
10. Lin, H., Raghavendra: A Dynamic Load-balancing Policy with a Central Job Dispatcher, IEEE Trans. on Software Engineering, 18(2), (1992), 148-158.
11. Feng, Y., Li, D., Wu, H., Zhang, Y.: A Dynamic Load Balancing Algorithm based on Distributed Database System, Proc. 8th Int. Conf. High Performance Computing in the Asia-Pasific Region, (2000), 949-952.
12. Genaud, S. et al.: Load-balancing Scatter Operations for Grid Computing, Parallel Computing, 30(8), (2004), 923-946.
13. David, R. et al.: Source Code Transformations Strategies to Load-Balance Grid Applications, LNCS, Springer Verlag, 2536, (2002), 82-87.
14. MPICH-G2: A Grid-enabled Implementation of the Message Passing Interface, Journal of Parallel and Distributed Computing, 63(5), (2003), 551 - 563.
15. Erciyes, K, Marshall, G.: A Cluster Based Hierarchical Routing Protocol for Mobile Networks, LNCS, Springer Verlag, 3045(3), (2004), 518-527.
16. Tunali, T, Erciyes,K., Soysert, Z.: A Hierarchical Fault-Tolerant Ring Protocol For A Distributed Real-Time System, Special issue of Parallel and Distributed Computing Practices on Parallel and Distributed Real-Time Systems, 2(1), (2000), 33-44.

Reconfigurable Scientific Applications on GRID Services

Jesper Andersson, Morgan Ericsson, and Welf Löwe

Software Tech. Group, MSI,
Växjö universitet,
351-95 Växjö, Sweden
{jesan, mogge, wlo}@msi.vxu.se

Abstract. This paper proposes a runtime environment for dynamically changing, parallel scientific applications. This kind of applications is motivated by the LOFAR/LOIS project aiming at a multidisciplinary research platform for natural scientists and engineers. The dynamic infrastructure in turn is than mapped to Grid Services environments.

1 Introduction

LOFAR (LOw Frequency ARray),[1] is a new generation, multi-purpose radio infrastructure aiming at multi-disciplinary research of astronomers, cosmologists, space and atmospheric physicists, climatologists, cosmic particle physicists, radio scientists, wireless communication developers and IT researchers. It consists of geographically digital receptor units connected to computing facilities with a high-speed network. Units are distributed over distances of 400 km; any unit will produce data at a rate of 2 Gbits/s, resulting in a total system data rate of 25 Tbits/s. The Swedish initiative *LOIS (LOFAR Outrigger In Scandinavia)*[2] aims, among others, at extending and enhancing the IT infrastructure capabilities of LOFAR.

The data received by the sensors are processed in scientific applications. Therefore, applications must be deployed to a computation infrastructure. Since applications can be deployed and/or removed dynamically and the computation infrastructure might change as well, runtime support for dynamic changes is required, as well. Such a runtime support functionality must perform the actual reconfigurations in a robust way. We propose a *dynamic service infrastructure* to solve these problems.

Grid Services have been established as runtime environments for scientific applications. The present paper evaluates how such established environments can be utilized to implement our new dynamic service infrastructure.

[1] http://www.lofar.org
[2] http://www.lois-space.org

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 813–822, 2005.

2 General Requirements of (LOIS-) Application

This section defines the concrete application environment that our research is embedded in. It defines some basic notions and roles, describes usage scenarios and, thereby, establishes requirements on a dynamic service architecture running on a Grid Service infrastructure. It also justifies our assumptions and constraints and serves as an example.

The kind of basic *hardware infrastructure* we are discussing consist of

- a number of sensors generating a stream of *input values*,
- a number of computation processors. These are referred to as *nodes*. Some nodes are connected to sensors receiving the input values. The latter are referred to as *input nodes*.
- an interconnection network allowing for communication between the nodes.

This is hardware infrastructure potentially managed by a Grid Service infrastructure.

On this Grid Service infrastructure, we execute the virtual experiments, i.e. applications processing data from the sensors. These *applications* are data parallel programs. Their input are either sensor input values or the output of other applications. If the input of an application a is the output of an application a', a is called *data-dependent* on a'. It is denoted by $a' \rightarrow a$. Applications are stateless, data driven, functions defined by *task graphs*.

The whole system to be executed is composed of a set of applications and their connections according to the data dependencies between them. The *configuration* of such a system is defined by:

- its set of applications,
- the data dependencies, and
- quality of services parameters

Since such a configuration itself defines a task graph, the whole system is an application, too.

Continuing, the configuration may change over time. These changes are triggered by different sources.

User Triggered: The *user* in our scenario is in charge of a certain experiment, i.e. controls a certain set of applications. User interactions are adding and removing applications. Adding new applications requires (i) connecting its input to the input values or to the output of running applications and, (ii) setting the quality of service parameters.

A typical scenario in this category is that a user adds a new experiment and necessary intermediate computations to the system and removes it after gaining the results.

Application Triggered: Some applications are detectors, recognizing certain patterns in the processed data that requires reconfiguration, for instance adding or removing applications and/or changing their quality of service parameters.

A typical scenario is the detection of an interesting sensor activity requiring an increased sampling rate. This leads to changed quality of service requirements and might lead to the situation where some on-line applications must be postponed and computed off-line. Their required data is then stored in the database, and their input is connected to a new application, a database daemon.

System Triggered: The complexity of applications might be input data dependent. Certain input might lead to load peaks in these applications. Also, hardware might fail. In order to guarantee the required quality of services, the runtime environment itself needs to reconfigure the applications.

A typical scenario is load balancing on sub-application level, i.e. redistributing some tasks in the data-parallel applications or postponing applications to off-line computations.

These changes are controlled by a dynamic service infrastructure, which is described in Section 3.1.

3 Architecture

This section describes the Dynamic Service Infrastructure (DSI), a reference architecture introducing dynamism to a static composition scenarios. It further shows how the DSI can be instantiated to solve the problem described in the previous sections. The DSI is further discussed in [3].

3.1 Dynamic Service Infrastructure – DSI

The DSI is a composite instantiation, influenced by architectural patterns proposed by [2]. The resulting architecture separates conceptual concerns from physical representation. The physical representation is the currently running system, while the conceptual architecture rather describes the structure and composition of the system. This conceptual architecture is needed in order to reason or reflect about the running system.

Conceptual Architecture. The conceptual architecture has two aspects: control and processing, as depicted in Figure 1. The control architecture is concerned with monitoring and, if needed, evolving the architecture to suit new requirements and/or constraints. The processing architecture captures the core behavior of a configuration of connected applications, i.e., it serves as a (high-level) description of the deployed system.

The **processing architecture** contains three basic component types; *Application*, *Connector*, and *Configuration*. These components and the relationships are depicted in figure 1. An *Application* is a container component where computations are performed. All interactions with a *Application* goes through a typed interface. A *Connector* captures component interactions. The typed end-points of a *Connector* connects to *Application* interface points. A *Configuration* is a composition of *Application* and *Connector* components. Some engineering tasks require hierarchical configurations where a *Configuration* contains other configurations.

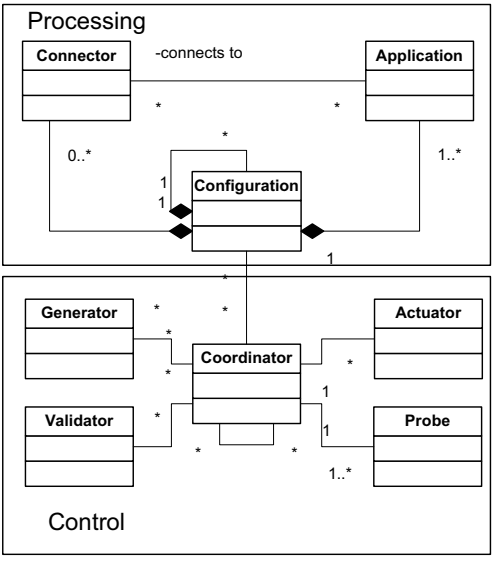


Fig. 1. Conceptual Architecture

The **control architecture** consist of five component types; *Probe*, *Actuator*, *Generator*, *Validator*, and *Coordinator*. The *Probe* and *Actuator* are the bridging points where control connects to processing. A *Probe* is a configurable monitoring process that monitor processing components and generate events that are communicated back to its Coordinator. The *Coordinator* component is the decision maker in the service architecture, responsible for coordination and delegation of control tasks. A *Coordinator* is goal-driven and can be employed on different levels in the architecture. The top-most *Coordinator* component strives for fulfilling the quality of service requirements posed by the different applications, while other *Coordinator* instances are responsible for coordination of sub-architecture (applications). In this process they employ different *Generator* and *Validator* components, for instance for creating new application configurations or rule-sets for coordination control and monitoring. The meta-data is provided by the processing architecture elements described above. The *Coordinator* use *Actuator* components to directly affect application configurations and/or application component instances.

This control architecture provide provisioning for the requirement discussed in previous sections. Reconfiguration and evolution is supported by *Coordinator*, *Generator*, and *Validator* components. For external initiation, (i.e Constructive Architecture style [2]), *Probe* and *Actuator* components are exported and included in external management applications. Application triggered reconfigurations are supported by the connecting *Probe* and *Actuator* components to *Application* and *Connector* entities in the processing architecture. For system level initiation, coordination is supported on different levels. *Coordinator*

components present in different applications connect to *Coordinator* entities on the system level. The network of *Probe* and *Actuator* instances also work on this level. Making these entities available for both on-line and off-line reconfiguration creates a highly flexible environment.

Physical Architecture. The principle tasks of the physical architecture is to deal with management of dynamic reconfigurations and events. In the most general case, the physical architecture can be responsible for the evolution of the system, i.e. the generation of new configurations on the fly. In most computation application, one could not expect a 1-to-1-mapping between the conceptual architecture and a working implementation architecture, which means that the generation of mappings turn into a costly activity. In our current setting this is not feasible, but we would not like to rule that out as impossible in the future.

In order to perform a robust, fine-grained deployment, the run-time coordinators must have access to meta-information describing the current conceptual level ant its mapped implementation. This information describe a *Configuration* of *Application* and *Connector* instances at different levels. The run-time system provide a modification language for the actual execution of a re-configuration of the implementation as well as a language for conceptual modifications. These includes primitives for the fundamental activities such as creation and connection.

The second assigned task is event management. Events generated externally (e.g a new schedule arrives) and internally (e.g increased sample rate) will be responded to proper actions taken. This is governed by a coordinator application. This coordinator is also responsible for feeding information back to the conceptual coordinator, for instance forwarding events that initiates a new lookahead schedule.

3.2 Mapping (LOIS-) Application to DSI

In order to use the DSI to run systems with requirements given in Section 2, it needs to be instantiated with a component and composition model, and with a strategy for mapping a conceptual to a physical architecture.

Components and Composition. A component is a data parallel synchronous program without any data distribution. For simplicity, it is assumed that the programs operate on a single array, a. The size of an input a, denoted by $|a|$, is the length of the input array a.

Array a is either the array of input values or the output of another component. Note that in our scenarios, the number of input values is fixed. We may assume that the output size of an application is a function of the input size. By induction, it follows that the input size is fixed for all applications in our systems.

A component can be modelled by a family of taskgraphs $G_x = (V_x, E_x, \tau_x)$ where τ_x is the execution time on the target machine. Scientific applications can automatically be compiled to such a family of task-graphs [10].

To this end, composition of components can be done by defining a program using these components and assigning the output array of one component to the

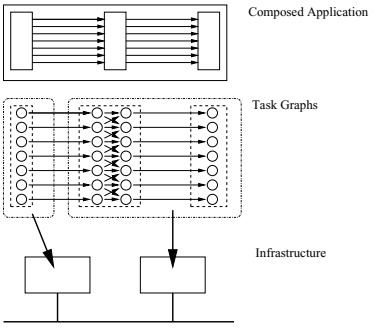


Fig. 2. Conceptual Architecture

input of another. This system is also a data-parallel program and can therefore be compiled and scheduled just like the individual components.

Mapping Conceptual to Physical Architecture. The process of mapping from conceptual to physical architecture is the process of translating the composed application to a family of task graphs and schedule these to the infrastructure. This process is triggered by certain events. Adding/removing an application/component requires a complete re-translation and re-scheduling

The processes of mapping from conceptual to physical architecture is a scheduling problem. This scheduling is triggered by certain events. Most of these events require a re-translation and a rescheduling and most be handled online.

Figure 2 sketches the complete process of mapping a simple, composed application to a task graph that is scheduled to two computational nodes.

In order to schedule a family of task graphs to the infrastructure a machine model to derive execution and communication costs from is needed. One such model is the LogP [4] model. In addition to the computation costs τ, it models communication costs with parameters Latency, $o$verhead, $g$ap, and the number of processors, P. At most $\lceil L/g \rceil$ messages can be in transmission on the network at any time.

The computation of an optimum scheduling, using LogP or simpler machine-models is known to be NP-hard. While there exists good approximations and heuristics, for instance [9], these are still to slow, given that certain events require online processing. One way of addressing this issue is to perform a lookahead scheduling [8], that is compute new schedules before they are actually needed. With lookahead scheduling, the frequency at which change events can be tolerated is the inverse of the delay for computing the lookahead schedules. This delay can be reduced by performing composition on task graph level instead of application level and using predefined schedules for the task graphs.

A task graph representation of an application/component is trivial to reuse, since the functionality of an application/component does not change. Given the nature of the composition it is also easy to do composition on task graph level.

A problem, however, is that while task graphs are trivial to reuse, the schedules of a task graph are not. Optimum schedules of individual task graphs (or their approximations) are, in general, not part of the optimal schedule for a composed system (or its approximation).

This problem is approached by modelling task graphs as *malleable tasks* and systems with *malleable task graphs*. A malleable task is a task that can be executed on $p = 1 \ldots P$ processors. Its execution time is described by a non-increasing function τ of the number of processors p actually used. For each task graph the schedules s_p can be pre-computed for $p = 1 \ldots P$ and $\tau(p) = TIME(s_p)$. A malleable task graph is recursively defined as a task graph over malleable tasks, i.e. nodes are ordinary task graphs or malleable task graphs and edges are the data-dependencies between them. How malleable task graphs can be scheduled is discussed in [8]. This scheduling needs only be done once for each malleable task graph.

4 Grid Services and DSI

This section shows how the DSI can be implemented using Grid computing. The section starts by defining a somewhat simplified, yet realistic Grid computing environment, and then shows how the DSI would map to it.

4.1 Grid Services

A grid is compromised of a wide array of heterogeneous resources. The role of the grid environment is to make sure that these resource interact and behave in a well-known and consistent way. This is achieved via a Grid framework. One such (conceptual) framework is the OGSA, the Open Grid Service Architecture [6], which brings service-orientation to the Grid world, by using an extended and refined Web Service model.

In order to implement the DSI using Grid computing some assumptions on the Grid framework are necessary. The major assumption is that there exists a central job management/load balancing service, that is replaceable. This service will be referred to as the coordinator service. It is further assumed that there is a replaceable factory service and some sort of packet management service. The Grid framework should also support the Web Services Notification Specification [7]. These assumptions are realistic, and all the required components can for instance be found in the gLite [1] grid middle-ware.

It is further assumed that a node represent a heterogeneous computational resource, that is anything from a sensor to a massive parallel computer, and that one grid service, representing a computational task, is running per node.

4.2 Mapping DSI to Grid Services

The OSGA specifies, as stated above, a number of services to manage the Grid environment and jobs executing within. While these services are not a 1:1 match with the components of the DSI, it is still possible to create a mapping.

According to the DSI, the processing architecture consists of a configuration and a set of applications and connectors. These applications are in turn described by a configuration and a set of connections and components (applications), in a recursive way. The applications and connectors resides in a packet manager service. This packet manager service can be seen as a more advanced version of for instance BSD ports collection or Gentoo Portage. Initially, the packet manager service has access to basic components matching typical operations found in a task graph. The set of connectors will vary a lot depending on the setting, but must include the required interprocess communication protocols, but local and remote. Examples could be shared memory, MPI, and so on. A SOAP/Grid Service connector must also be included to allow for message passing between different computational nodes. A skeleton Grid service component, implementing the basic WS-Notification, life-cycle management, and so on, must also be included. The set of components will grow with every requested composite component.

The main component of the control architecture is the coordinator, which is charge of the scheduling and deployment. This is typically the job of any job/task management service or load balancing service in the Grid environment. Any such service with a reasonable API should be able to be replaced by the coordinator service. The generator and validator components are also part of such a job manager.

The probe and actuator are implemented as part of the Grid services. This can be coupled with the event management system, implemented using WS-Notification.

A shown in Section 3.2, the scheduling of an application to an infrastructure creates clusters of tasks to be executed on a node by decomposing an application according to quality of service and data constraints. Every such cluster of tasks is represented by one Grid service. The creation of such a cluster is handled by the factory and packet manager services, and initiated by the Coordinator service.

4.3 Reconfiguration

As stated above, the job of the scheduler is to partition the task graph to computational nodes, that is create a set of Grid services. This is done by the scheduler creating a configuration and deliver it to the factor service. The configuration contains a set of services mapped to nodes, where each service is described as a set of components and connectors. The factory queries the packet manager for the components needed and deploys the composed pieces of software on the assigned nodes. The packet manager caches all composed services and keeps track of all running instances of a service, in order to avoid re-deployment of identical services.

The scheduling remains unchanged from the general LOIS case. We do assume that the entire system is rescheduled without consideration of running system. It would be possible to use heuristics in form cost penalties when switching communication-type to encourage the use of an existing configuration. The

scheduling will be more complex given the different layers, that is the scheduling of tasks within a node could be just as complex as the scheduling of task clusters to nodes. Another issue is that a machine model describing the Grid environment must be developed. [5] presents a experimental study of LogP parameters using the Web Service model. This work can easily be extended to suit a Grid scenario.

5 Conclusions and Future Work

This paper discussed the application scenario of LOIS, a radio and data network for scientific applications, i.e. systems of data-parallel applications requiring high performance. Additionally, applications could be added/removed dynamically. In our scenario, the system architecture could even change due to the results of applications. A solution to address both the performance and the flexibility problem is the Dynamic Service Infrastructure (DSI) proposed in this paper and its implementation on top of a Grid Services environment.

The implementation of the DSI on top of a Grid environment is an interesting result since it, at least theoretically, shows how LOIS could utilize Grid resources and contribute to existing Grid resources. Another result is that the dynamic properties of the DSI can be reused by any Grid application as long as the basic assumptions regarding applications and development models hold.

Much work remains to be done. The results presented here are mostly theoretical. A implementation of the components described here is underway, even if not all of them are currently in a Grid environment.

Another problem is the scheduling and machine models for Grid services. Work is ongoing on defining a multi-paradigm scheduling policy that can handle the scheduling required, and initial experiments are carried out to define a machine model for Grid/Web services that is both simple and realistic enough.

References

1. EGEE middleware architecture. Technical report, Enabling Grid for e-Science in Europe, 2004.
2. J. Andersson. A classification of dynamic software architectures. Technical report, Department of Computer Science, Växjö universitet, 2003.
3. J. Andersson, M. Ericsson, and W. Löwe. An adaptivehigh-performance service architecture. In *Software Composition Workshop (SC) at ETAPS'04*. Electronic Notes in Theoretical Computer Science (ENTCS), 2004.
4. D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP 93)*, pages 1–12, 1993. published in: SIGPLAN Notices (28) 7.
5. Morgan Ericsson. Web Services and the LogP machine model – An experimental study. In *3nd Nordic Conference on Web Services*, 2004.
6. I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The physiology of the grid. Technical report, The Globus Alliance (www.globus.org), 2002.

7. S. Graham and P. Niblett. The web-services notification specification 1.0. Technical report, Akamai Technologies, Computer Associates International, Fujitsu Laboratories of Europe, Globus, Hewlett-Packard, IBM, SAP AG, Sonic Software, TIBCO Software, 2002.
8. W. Löwe J. Andersson, M. Ericsson and W. Zimmermann. Lookahead Scheduling for Reconfigurable GRID Systems. In *10th International Euro-Par Conference*, 2004.
9. W. Löwe and W. Zimmermann. Scheduling balanced task-graphs to logp-machines. *Parallel Computing*, 26(9):1083–1108, 2000.
10. Wolf Zimmermann and Welf Löwe. An approach to machine-independent parallel programming. In *CONPAR '94, Parallel Processing*, volume 854 of *LNCS*, pages 277–288. Springer, 1994.

Geographic Information Systems Grid*

Dan Feng[1], Lingfang Zeng[1], Fang Wang[1],
Degang Liu[2], Fayong Zhang[2], Lingjun Qin[1], and Qun Liu[1]

[1] Key Laboratory of Data Storage System, Ministry of Education,
School of Computer, Huazhong University of Science and Technology, Wuhan, China
[2] GIS Software Development & Application Research Centre, Ministry of Education,
Info-engineering College of China University of Geoscience, Wuhan, China
dfeng@hust.edu.cn, zenglingfang@tom.com

Abstract. GIS Grid (geographic information systems grid) is a combination of geographic information systems and grid technology. Existing non-standardized multi-sources and multi-scales data have a shortage of spatial information shared in either internal and external organizations or departments, especially in national or global applications. It is very important and difficult to store massive spatial data for GIS Grid. And efficient metadata management and data access methods should be studied. At the same time, GIS Grid should provide users with just the services and data they need, without having to install, learn, or pay for any unused functionalities. With the integration of web service and other middleware/toolkit, which support the creation of Grid infrastructures and applications, it is possible to dynamically assemble applications from GIS Grid for use in a variety of client applications.

1 Introduction

In the past forty years, although fast development of geographical information and computer-science integrated technology demonstrates that spatial information is widely used in people's everyday lives. GIS (geographic information systems) [1,2,3] is important aspects of spatial data infrastructure, and it also is a powerful, advanced, intelligent, integrated software system. GIS systematically combines such sciences as graphics, image, geology, geography, remote sensing, mapping, artificial intelligence and computer science. GIS are used in many fields, including geology, petroleum, ocean, remote sensing, urban and regional planning, cadastral registration, water supply systems, communication facility management, gas, transportation, electric power, agriculture and irrigation works, fire and epidemic control, land resources, disaster monitoring, regional crops, trading, transportation, tourism, ecology and

* This paper is supported at Huazhong University of Science and Technology by the National Basic Research Program of China (973 Program) under Grant No. 2004CB318201, National Science Foundation of China No.60273074, No.60303032, Huo Yingdong Education Foundation No.91068 and at China University of Geoscience by High Technology Research and Development Program of China No.8001AA135170, plus industrial support from WUHAN ZONDY INFO-ENGINEERING CO., LTD.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 823 – 830, 2005.

environment and population subsystems. The role of GIS in these applications is to provide the users and decision makers with effective tools for solving the complex and usually ill- or semi-structured spatial problems. The computer-based systems are designed to support the capture, management, manipulation, analysis, modeling, manipulation, retrieval, analysis, and display of spatially referenced data. But, traditional GIS is used in the limitative organizations or limited region based on many kinds of systems (e.g. AUTOCAD, ARC/INFO, MAPINFO, MAPGIS etc.), and spatial data, come from multi-sources and multi-scales (e.g. remote sensing, electronic thermometer total station, global position system etc.), lack for collaborations and resource sharing among different organizations or regions.

Grid [4,5], is a new class of infrastructure, promises to make it possible for public application collaborations to share resources on an unprecedented scale, and for geographically distributed groups to work together or individual to conveniently live. Grid coordinates resources that are not subject to centralized control and uses standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service (QoS). Grids have moved from the obscurely academic to the highly popular, such as Compute Grids, Data Grids [6], Science Grids, Knowledge Grids, Commodity Grids, Bio Grids, Resource Grid, Service Grid, Information Grid and Sensor Grids.

GIS Grid integrates the traditional GIS software, database software and data analysis software into an ideal platform. Therefore, GIS Grid creates an ideal environment for the comprehensive assessment and analysis of multi-sources geology data. It is mainly used by regional planners, managers and general users. GIS Grid integrates a series of application systems, such as urban planning, communication network management, water supply management, gas network management, municipal management, power network management, cadastre management, land use programming, land information management, global position system (GPS) navigation and monitor, environment protection and supervision.

2 Spatial Object-Based Storage

Spatial information resources such as remote sensing, digital maps, and paper maps are collected from different organizations or departments. Statistics data, including agriculture, forest, earthquake hazard, ore, hydrogeology, ecology, transportation, population, tourism, and trade, are the composition of different application systems. Spatial data are multi-terabyte and require massive storage system. The architecture of spatial data is objects, fundamental containers that house both application data and an extensible set of object attributes. Traditional storage systems, such as DAS, NAS and SAN, are unsatisfied with these requirements.

Fortunately, for GIS Grid, new storage architecture is emerging. OBS (object-based storage) [7,14,17] is the foundation for building massively parallel storage systems that leverage commodity processing, networking, and storage components to deliver unprecedented scalability and aggregate throughput in a cost-effective and manageable package. And OBS can be extended in WAN (wide area network) for its high scalability (Figure 1).

In OBS, spatial objects are decomposed into a set of storage objects and distributed across one or more OBSDs (Object-based Storage Devices) [15,16]. Each OBSD includes local processing capabilities, local memory for data and attribute caching, and its own network connection. And much of the traditional storage allocation activity can be offloaded from the file system layer. So a key performance bottleneck present in current storage systems (e.g. NAS) is removed.

With object attribute scalability, abundance clues are obtained to guide or direct in the solution of active application (AA), and AA realizes by dynamic loading/unloading some method using JVM (java virtual machine) technology. AA has a method pool filled with active application methods regarded to be useful by one or more of the management policies. Most popular methods such as read with filtering, write with encryption and hot spot migration are initially registered to the method pool. AA also has a policy pool. A new policy registered into the policy pool has to contact corresponding method in case they are not already registered. AA ensures that the registered methods are called by storage system according to its environment. Those policies themselves are just descriptions of how to implement system management function and specify system states and how to response to them. For GIS Grid, those policies will ensure that the best ones for the current workload or performance are triggered. AA triggers the policy depending on statistics, which are useful for enforcing credential-based access and QoS policies, and supporting dynamic data redistribution for cross-OBSD load balancing. OBS mirrors the scale-out architecture of GIS Grid, providing a balanced growth model that adds network bandwidth and processing capability in step with capacity increments to ensure scalability.

3 Metadata Management and Spatial Data Access

In GIS Grid, user's application may be very complex. For example, a decision-maker of a city, he must acquire the assistant information offered by the planning organization and other organizations, such as economic developing zone, urban planning bureau, land administration bureau, academy of planning & surveying, archives of urban construction, water supply general company, gas company, electric power company and telecom general company. Taking the city's geographical map as background, the decision-making process must be supported by GIS Grid.

3.1 Metadata Management in GIS Grid

Massive spatial information system design technology is complicated work based on different information sources and demands. For GIS Grid, spatial data standards are very important for system establishment, which may result in considerable duplication or hard data transfer burden. Existent spatial data standards and drafts (e.g. ISO/TC 211, FGDC) are widely used in spatial metadata management.

Figure 1 shows several types of metadata [10] that may be used to manage spatial objects. At the OBSD level, physical metadata includes information about the characteristics of objects on physical storage device, which is inode-like information. Based on physical metadata, OBSD provides object interface, which is different from

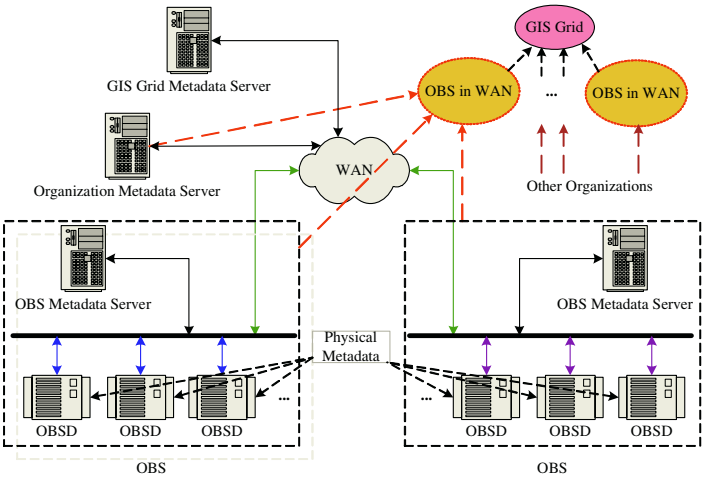


Fig. 1. Massive spatial data management in GIS Grid

traditional file systems or database management systems. OBS metadata is in the second level. Object IDs are identifiers for object data that may be striped in several OBSDs in one OBS. And OBS metadata is often organized in DFS (distributed file system). Organization metadata is above the OBS level, and is managed by LDAP (lightweight directory access protocol). The top level is GIS grid metadata, which is managed by database with powerful queries.

3.2 Spatial Data Access for GIS Grid

Spatial object access is the process of identifying spatial objects of interest to the user. MS (metadata services) [8,9,10] is defined as services that maintain mappings between Object ID and object storage, and respond to queries about those mappings for above-mentioned four levels. MS play a key role in the publication and the discovery and access of spatial object. Figure 2 shows a modified version application services originally given in [11] by Alameh. Spatial data are based on the value of descriptive attributes. Typically, MS forms one component of a series of object accesses in a GIS grid.

In figure 2, *Portrayal service* [12] assembles and portrays an ortho-image from several imagery access services. And to integrate other spatial information, it provides interfaces for Web Services to implement specified spatial service requirements. *Portrayal service* also is used to deal with spatial information synthetically according to users' requirements and their characters. *Web mapping service* (WMS) provided by distributed places and GIS manufacturers, such as ESRI's ArcIMS service, MapInfo's MapXtreme service, MapGIS IMS, returns a map corresponding to pre-specified rectangular geographic extent and pixel dimensions for a given area. *Web mapping service* is required by portrayal service or integrative service. It also provides services for mobile clients, e.g. LBS (location based service) and GPS services. *Integrative services*, which multi-party providers usually supply, bundle predefined services and

present them to the client as one. *Integrative services* perfectly join MIS (management information system) or OA (office automation) with GIS, which integrate the transactions of design, construction, maintenance and professional analysis of many organizations or departments and provide the related branches with professional, multi-demand and multi-aim services. Those disposal also include making and editing legend of thematic map, automatic topological maintenance, routes analyzing, cash analyzing, cutting down, three-dimension analyzing, statistical analyses and so on.

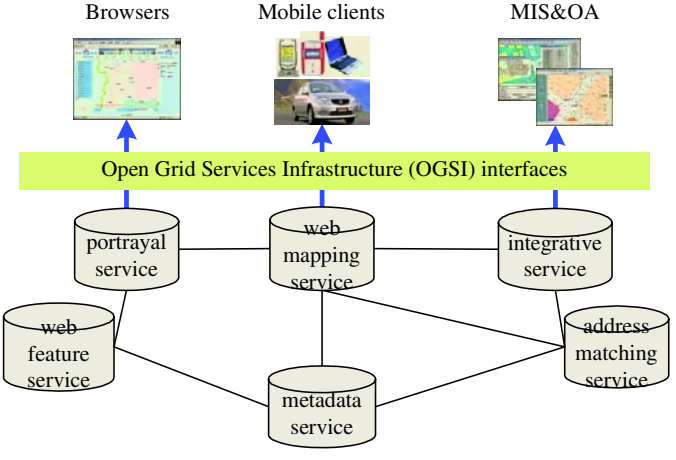


Fig. 2. Different services in GIS Grid

There are some scenarios for LBS and GPS service in GIS Grid. Some people want to travel from one place to another by driving bus themselves. There is only one thing to do: input their whither. GIS Grid first provide web mapping service. With help of metadata service, address matching service optimize and list all the cities, towns, roads and mileages etc. All that will be carried out by their cars' GPS. Specially, integrative service offer all concerned hotels, travel industry sites, sceneries' introductions etc. Also GIS Grid gives their some references about the journey cost. Here, objects are a series of site entities which are concerned geographical information. And these distributed objects are stored in OBSDs.

4 GIS Grid Services

Using international and national standards or drafts will be a popular way for GIS Grid forming. Table 1 shows GIS Grid services protocol layers and related standards [12]. Open GIS Consortium (OGC) sponsors consensus-based development of interoperable GIS web services interfaces. Open Grid Service Infrastructure (OGSI) defines web services. Every Grid service is a web service [13]; it uses Web Service Definition Language (WSDL) as the mechanism to describe the public interfaces of Grid services.

GIS Grid services follow OpenGIS Web Services Architecture (OWSA) [12] and OGSI for creating, managing, and exchanging information among organizations. GIS Grid services also provide human interaction to manage user interfaces, graphics, and to present compound documents. The development, manipulation, and storage of metadata, conceptual schemas, and datasets are managed by information management service. Specific tasks or work-related activities are supported by workflow services. Processing services perform large-scale computations. Communication services encode and transfers spatial data across networks and system management services manage system components, applications, networks and access control. Therefore, it is easy to integrate existing non-standardized multi-sources and multi-scales data (Users need not to know the details of their realization and the platforms they run) with the developments according to GIS Grid services. Now most developing tools which support those services protocol stack, particularly, the Java 2 Platform, Enterprise Edition (J2EE), built by Sun Microsystems and other industry players, and Microsoft .NET, built by Microsoft Corporation all are based on Web Service.

Table 1. GIS Grid services protocol stack

Protocol layers	Dominant protocol standards
Service integration & workflow	WSFL etc.
Service discovery	UDDI, OGC-Catalog etc.
Service description	OGSI, OWSA, WSDL
Service	HTTP, SOAP, COM, CORBA, J2EE, SQL
Binding	OGC SF, Coverage, Coordinate transform, WMS
Data format, schema and semantics	HTML, OGC-GML, OGC-WKT/WKB, XML/S
Data representation & encoding	ASCII, XML, ASN.1/DER
Communication protocols	TCP/IP, FTP, SSL, SMTP

GIS Grid service work like the following descriptions and require in point technologies: As provider, GIS Grid creates, assembles, and deploys services using the programming language, middleware, and platform of the provider's own choice. GIS Grid service defines the web service in WSDL. Web service needs to invoke other heterogeneous distributed web geographic information systems (WebGIS). WebGIS provide application services to different operations. GIS Grid interface registers the service in Universal Description, Discovery, and Integration (UDDI) registries. GIS Grid interface deals with users' require, if the require relate to spatial information, spatial service or other spatial application system function, it calls related functions or services to handle it. UDDI enables developers to publish web services and that facilitates their software to search for services offered by others. Prospective users find the service by searching a UDDI registry. These users' applications bind to web service and invoke the service's operations using Simple Object Access Protocol (SOAP). SOAP offers an XML format for representing parameters and return values over HTTP. It is the communications protocol that all web services use.

5 Conclusion

85% data are related with geographic information in the world. Today, digital earth is on the way. A concept of GIS Grid is first presented in this paper. The storage and access mode about spatial object is provided. And Grid GIS service is also studied by contrasting and utilizing OGSI. With the help of grid technology, GIS is evolved from the traditional model of single application, in which spatial data is tightly coupled with the system used to create them, to an increasingly distributed model based on independently provided, specialized, interoperable Grid web services. This evolution will be fueled by factors such as GIS Grid's growing role in future, spatial data's increasing availability and inherent conduciveness to reuse.

Reference

1. "GIS Standards and Standardization: A Handbook". United Nations Economic and Social Commission for Asia and Pacific, New York (1998)
2. Shanzhen Yi, Lizhu Zhou, Chunxiao Xing, Qilun Liu, Yong Zhang. "Semantic and interoperable WebGIS". Proceedings of the Second International Conference on Web Information Systems Engineering (2001) 42-47
3. Yongping Zhao, D.A. Clausi, "Design and establishment of multi-scale spatial information system based on regional planning and decision making". Geoscience and Remote Sensing Symposium, 2001. IGARSS '01 (2001) 1965-1967
4. I. Foster, "What is the Grid? A Three Point Checklist". GRID Today, July 20, 2002. Access from http://www.globus.org/research/papers.html
5. I. Foster, "The Grid: A New Infrastructure for 21st Century Science". Physics Today, (2002) 42-47
6. The CERN DataGrid Project, http://www.cern.ch/grid/
7. Intel Corporation, "Object-Based Storage: The Next Wave of Storage Technology and Devices", January 2004, accessible from http://www.intel.com/labs/storage/osd/
8. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets". Journal of Network and Computer Applications (2001) 187-200
9. S.A. Brandt, E.L. Miller, D.D.E.Long, Lan Xue. "Efficient metadata management in large distributed storage systems". 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (2003) 290-298
10. G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Mahohar, S. Pail, L. Pearlman. "A Metadata Catalog Service for Data Intensive Applications". Proceedings of Supercomputing 2003 (SC2003)
11. N. Alameh. "Chaining geographic information Web services". Internet Computing (2003) 22-29
12. "OpenGIS Web Services Architecture", document OGC 03-025, Open GIS Consortium, Jan. 2003, access from: http:// http://www.opengis.org/
13. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling. "Open Grid Services Infrastructure (OGSI) Version 1.0". Global Grid Forum Draft Recommendation (2003)
14. M. Mesnier, G.R. Ganger, and E. Riedel, "Object-based storage", Communications Magazine, IEEE, Vol 41, Issue: 8, pp. 84–90, Aug. 2003.

15. SNIA, Object-Based Storage Devices (OSD) workgroup, January 2004, accessible from http://www.snia.org/osd
16. Dan Feng, Ling-jun Qin, Ling-Fang Zeng, Qun Liu, "a Scalable Object-based Intelligent Storage Device", Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, 26-29 August 2004,pp 387-391
17. Ling-Fang Zeng, Dan Feng, Ling-jun Qin, "SOSS: Smart Object-based Storage System", Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, 26-29 August 2004,pp 3263-3266

Tools for Distributed Development and Deployment on the Grid

Ariel García, Marcus Hardt, and Harald Kornmayer

Forschungszentrum Karlsruhe, Institut für wissenschaftliches Rechnen,
Postfach 3640, 76021 Karlsruhe, Germany
{garcia, hardt, kornmayer}@iwr.fzk.de

Abstract. The development and deployment of middleware and applications in a grid environment spread over many institutions is a complex challenge. The management of the development process is of key importance for the successful completion of the software components. This paper presents the measures taken and the experience gained within the EU project CrossGrid. After briefly presenting the infrastructure, the development procedures and the supporting tools are described in detail. The CrossGrid "common directory" approach allows an effective, centrally managed configuration of distributed resources. The experience of the CrossGrid integration team is discussed.

1 Introduction

The integrated and collaborative usage of a distributed infrastructure of computers, networks, and potentially also scientific instruments is nowadays called a Grid [1]. Thereby Grids are different from local clusters because the connected resources are owned and managed by multiple different organizations. Furthermore, not only the resources are distributed, but also the the developers in most Grid projects are spread over several institutions in different countries. These border conditions make the development process in a Grid project challenging concerning the management of the software development, its deployment and the maintenance of the distributed system.

The standard procedure of Grid middleware development starts with software development on local resources (workstations, clusters), including local tests. Afterwards, the software needs to be deployed on the distributed infrastructure for testing. The test conditions are not managed centrally and they will be influenced by changes on the distributed resources, since they are under the control of local site and network administrators and their policies. For this reason testing is a complex problem. After testing, the software has to be deployed on the production system. But even if there is no central control of the resources, all steps of the development process must be coordinated and managed. It is important for an effective development that all these steps have a short time scale.

The EU funded project CrossGrid [2] ports four societally important, novel applications to a Grid environment. These applications require interactivity from

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 831–839, 2005.

the Grid middleware. Therefore CrossGrid also develops a set of middleware tools that allow a human to interact frequently with the applications. All the development is done on a testbed spread over 9 European countries, including new member states. This paper describes the development process in CrossGrid. Additionally, it presents all the tools developed and used by the CrossGrid consortium to support the development, deployment and maintenance of the software on the project's testbeds.

2 The CrossGrid Testbed

CrossGrid is developing interactive Grid applications in the domains of medical surgery planning, flood crises support systems, high energy physics analysis and environmental prediction systems. All CrossGrid applications use the Message Passing Interface (MPI) standard [3]. Moreover, CrossGrid provides Grid development tools and Grid middleware components to ease the migration of applications to a Grid environment.

The infrastructure for these developments is distributed across 16 sites in 9 countries [4]. All these sites are running the CrossGrid middleware, which is based on the middleware from the European Data Grid (EDG) project [5], currently taken over by the LHC Computing Grid [6] and called LCG-2. It is based on the Globus Toolkit [7] version 2 middleware. Most sites use the Linux distribution Red Hat 7.3, except one which is using Mandrake 8. As usual in grids running EDG-derived middleware, the services present at each site are the Compute Element (CE), the Storage Element (SE), the User Interface (UI) and a number of Worker Nodes (WN). For the installation and configuration an LCFG [8] server is used. Additionally, several central servers are required. The Replica Location Service (RLS) takes care of the data management, while the VO server is needed for user authorization. The Resource Broker (RB) schedules job requests based on resources' status information from the Information Catalogue (IC) to sites that fit the jobs' description. This Resource Broker contains CrossGrid modifications which make it capable of submitting MPI jobs to multiple CPUs at one site (MPICH-P4) or spread over many sites (MPICH-G2). CrossGrid provides an easy access to the grid via the Roaming Access Server (RAS), which serves a java applet able to run on any client machine. This java applet is called the *Migrating Desktop* (MD), and communicates with the RAS server in order to access the Grid. Interactive steering of jobs is done via the so called Job Submission Services (JSS), which are also deployed inside the RAS machine.

At the beginning of the project, a big effort was invested in setting up testbeds and all the supporting tools. A separate testbed was used to evaluate preliminary middleware releases. The resources of this evaluation testbed were added to the production testbed once a stable middleware emerged. After the successful installation of the LCG-1 release, it was recognized that development of middleware and applications on the same testbed introduced too many side effects, and was slowing down the development process. Every test of a new middleware component affected the stability of the testbed, thus conflicting with application development.

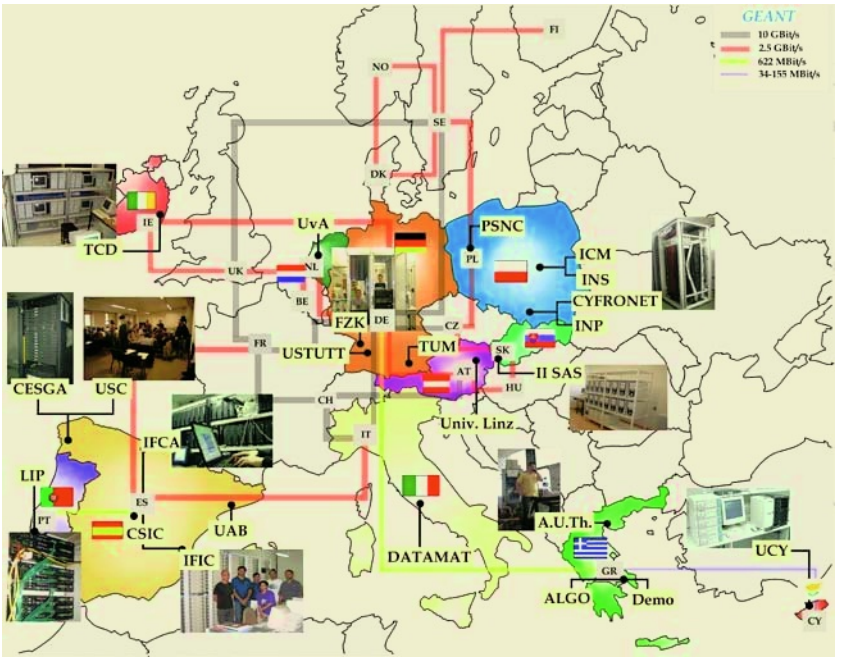


Fig. 1. The CrossGrid testbed

In order to separate both kinds of development and to decrease the time of the middleware improvement cycle, the CrossGrid testbed was split in two. One was focused on fast development, deployment and testing of middleware, the other on offering production level services for application developers. A *virtual Chinese wall* between them was introduced. Therefore, a second instance of each of the central services had to be installed. Every site is forced to live on only one side of this Chinese Wall.

The *development testbed* consists of 5 sites supporting fast middleware deployment and testing. The effort imposed on a developer for getting a new package into this testbed is kept as low as possible. I.e., only packaging and basic requirements must be fulfilled. This enables a short deployment cycle of down to a couple of hours.

The *production testbed* is intended to be more stable. All middleware components have to be tested and validated prior to their deployment. This testbed is supported by 13 sites and is therefore more powerful.

3 Distributed Development

All 21 partners of the CrossGrid project take part in the development process. A common grid system can only be achieved by coordinated efforts. Clear and

comprehensible development guidelines are as essential as an organized communication between different members of the project. The tools used within CrossGrid to support the development process and to ensure a proper software quality are described in this chapter.

3.1 Development Rules

To control the divergent forces in a heterogeneous project like CrossGrid a comprehensive set of guidelines for developers are of outstanding importance. Therefore the Developers' Guide from the EDG project [5] has been adapted and modified to fulfill the needs of CrossGrid. The CrossGrid Developers' Guides [9, 10] define the processes and the responsibilities within the whole development process, covering the phases of application development, coding style, testing, deployment, integration and release.

3.2 Communication Tools

During the project different communication tools were used. At the beginning of the project, managed mailing lists where set up for organizational issues using the Majordomo [11] tool. But the utility of these lists was limited to distributing managerial information only. The audience of the lists was too big to discuss all the technical details of the project. To allow the developers to setup their task-specific mailing lists, the GridPortal server [12] – based on the Savannah software [13] – was used.

Developers at different sites autonomously started to use instant messaging tools of various flavours to solve problems during the integration of different components. No central control was established to ensure the availability of all CrossGrid developers. This resulted in small groups of developers using instant messaging tools, although the utility was quite high.

From the beginning of the project the freely available VRVS system [14] was used for running audio/video conferences. These were necessary and useful for having a broader discussion of all aspects of the project. When the integration of all components to a common system became the most important issue, weekly VRVS integration meetings were scheduled to report on the status of the software components, the testbed sites and the releases. These audio/video conferences helped in avoiding problems between different software parts in advance. Communication continued on a dedicated mailing list.

3.3 Development Tools

Because of the many different programming languages used in CrossGrid no common development tool was recommended. But to ensure a collaborative and distributed development some central resources like a software repository, a bug tracking system and a standard build environment for the software had to be provided.

The GridPortal server [12] is offering some of these central development services. It is accessible via a standard web browser. Organized in projects or small groups of developers the following tools were used by CrossGrid:

CVS: The Concurrent Versions System [15] is a client-server based versioning tool and is used as the central software repository of all CrossGrid components. CVS allows distributed development to ensure different developers can work on the same source code at the same time. In the case of a conflict the developers are forced to solve it before the changes are accepted.

Bug Tracker: The Savannah bug tracker allows any user to submit bug reports to the developers of the corresponding software component by using the "trouble ticket" metaphor. Everybody involved is informed by email about changes to the ticket. If the problem remains unsolved for more than 30 days an email reminder is sent to all involved project members. This tool proved to be useful during the integration and testing phase of the components.

Webspace: To exchange additional information about a software component, the GridPortal server provides webspace for publishing documents and binary packages.

The standard build environment of CrossGrid is based on the autobuild [16] tool developed by the EDG project. It takes the source code of every package out of the software repository and runs a standardized build procedure to produce installable packages (tar.gz, src.tar.gz, rpm and srpm). A build failure results in a notification to the developers. The build environment is centrally managed and based on a standard testbed node. This prevents undocumented modifications of the build procedure. The build process is started every 4 hours and the results are available on a GridPortal web page. Many software dependencies were found by the autobuild procedure. This resulted in an improvement of the software quality.

4 Deployment

Deployment of a complex software system like the CrossGrid middleware and tools on a heterogeneous system requires a fast and lightweight roll-out process. The CrossGrid software is available as rpm packages [17] from the GridPortal server. Not only the download of the software packages, but also the site configuration must be as automated as possible. Additionally, the software versions of the development and the production testbeds will be different. In this chapter the CrossGrid configuration and deployment approach is described in detail.

4.1 Testbed Configuration

To automate the configuration as much as possible, and to control the installed releases at all sites, CrossGrid uses the LCFG [8] installation and configuration tool. The configuration of each site is split in two parts – a site dependent part and a part common to all sites.

The site dependent configuration contains local parameters like IP-addresses, etc., and is derived from common template files.

The common CrossGrid configuration contains the software package lists and the configuration that is required by the middleware. It is mandatory to be used at every site, and it is kept inside the so called *common directory*.

Since CrossGrid is based on the EDG middleware, which is evolving inside the LCG project, it has to follow the changes of the underlying middleware. To manage the development within CrossGrid, the common directory approach was extended. A copy of the LCG-configuration files was placed into the LCG subdirectory of the common directory. Wherever this configuration needed to be altered, it was overwritten by an appropriate file inside the CrossGrid subdirectory of the common directory. CrossGrid sites use the include-mechanism of the LCFG tool in order to accomplish this.

For managing the differences between the CrossGrid development and production testbeds, CrossGrid uses different CVS branches of the same files. This allows an easy migration of tested software from the development to the production testbed.

The CrossGrid common directory configuration approach has the following advantages:

- The same set of basic configuration files is used at every site.
- The difference between *CrossGrid base* and *local* configuration at a given site is clearly documented.
- The ongoing development of the "EDG/LCG Middleware" can be coped with by mirroring their configuration into the CrossGrid common directory in CVS.
- A CrossGrid version can be trivially defined by putting a CVS tag on the common directory.

For deployment CrossGrid uses only binary rpm packages, because all hardware components are i386 based. These packages are autobuilt and published on GridPortal. To automate the deployment process, CrossGrid developed the tool `cg-lcfg-go` [18] for site administrators. It provides the following functionality:

- update the configuration files from CVS on the LCFG server.
- download missing RPMs to the local package cache.
- install and update required RPMs on the LCFG server.
- compile profiles for all managed clients.

This tool substantially eases the update and maintenance work, allowing a site to update to a new release in just a few minutes with high reliability.

Although the deployment process was highly automated, the CrossGrid integration team recognized that it is unavoidable to control the software that is actually installed at each site. The presentation of the current site status is essential and a webpage automatically publishing this information was installed in the GridPortal. Additionally, a tool to monitor the differences between the installed and required packages for every testbed node was developed.

4.2 Test and Validation Procedure

The CrossGrid testbed was split in two parts in order to get one environment for development and one stable environment for running and developing applications. To ensure good software quality on the production testbed, a procedure for testing and validating software components was introduced [19]. Each software component that needs to be installed on the production testbed must successfully pass this process. This is an important part of the whole development and integration process of CrossGrid.

To migrate a software component from the development to the production testbed, developers ask for a test and validation process. Information like dependencies, configuration, installation files, network requirements and behaviour of the software must be provided. The test and validation request is assigned to a test person, who installs the software at his site. After installation he tests the functionality of the software and the stability of his site. If problems occur they are reported in the bugtracker (Section 3.3). Depending on the severity of the problems which turned up, the test request can pass, pass with minor issues, be rejected with major issues or completely fail. If the package passed the procedure, the software is allowed to be installed on the production testbed. This procedure minimizes the probability of an unstable production testbed.

5 Discussion

The CrossGrid project offered the chance to collect experiences in development of grid middleware, the deployment in a complex grid environment and the maintenance of a distributed testbed. In this section the lessons learned will be discussed.

5.1 Developers Guide

The maintenance of a distributed system like the CrossGrid environment takes some effort, especially when the integration of different middleware components and applications starts. The developers of the components need to have a common view of the system to reduce the number of side effects to a minimum. Therefore the knowledge of the testbed setup and deployment procedures is essential. The developers must understand their limitations in a distributed system as compared to their local cluster. It is of the same importance, that the site administrators and integration team members provide a lightweight infrastructure to support the development and the testing of software components. All the development cycles must be as short as possible.

Therefore developers need to have a short and easy to read Developers Guide, which describes the development, the packaging and the installation of software on a testbed. Also the testing and the quality assurance methods must be transparent. The CrossGrid Developers Guide is a collection of best practices, with experience gained by EDG and CrossGrid. In each project such a Guide must

be published as early as possible and updated frequently in order to prevent the development of undeployable software.

Centrally managed guidelines make the development of Grid middleware and applications more effective. The release and deployment of software within CrossGrid was much easier after those guidelines were accepted by all project members.

5.2 Deployment

The management of the testbed was very important during the development and testing of CrossGrid middleware and applications. Dividing the testbed in two parts and introducing a standardized Test and Validation procedure increased the overhead for the developers, but after the acceptance of both testbeds and the new procedures development and testing was much easier and better organized.

The dedicated autobuild system proved to be useful to provide the developers with a fast response time regarding packaging and deployment.

The introduction of a fast deployment method made the whole development more effective. With the common directory approach CrossGrid provided a mechanism to keep a centralized control of the software to be installed in a distributed testbed. The installation of a new release can be performed in a minimal time.

The CrossGrid integration team benefited a lot from the tools developed for controlling the deployed software at different sites. The webpage showing the current status of each site has positively influenced the deployment speed of the testbed releases. This, together with the tools developed for simplifying the testbed upgrades, was the base to reducing the deployment time from around 10 days to one day. This deployment time was achieved in the CrossGrid development testbed, where site administrators are aware of the changes, and release cycles happen more often. The deployment on the production takes a little longer – up to 3 days. The grid deployment is controlled by the integration team. Sites not updating within one week are removed from the CrossGrid environment and mentioned on the websites. This occurred rarely and was related to a temporary lack of manpower at the site.

6 Conclusion

Grids are distributed environments overcoming institutional and enterprise borders. Similarly, the development of middleware and applications is often done in developer teams spread over different locations. The development process must take this into account by providing clear guidelines for developers and a centralized, managed packaging and deployment process. The organization of the development and the maintenance of a grid system is of essential importance for success. Effective tools for development, deployment and monitoring are available from grid projects like CrossGrid.

Grid systems are not self-organizing. They need to be developed and tested under managed, well defined conditions. A big effort is necessary to ensure that

all developers and administrators follow these guidelines. Coordinated communication in a distributed grid environment needs a sufficient integration team with full support from all partners including the project management.

Acknowledgement

The authors thank all members of the CrossGrid project, and especially Jesus Marco, Jorge Gomes and Mario David for their support. Further thanks to the think tank of Workpackage 7.

CrossGrid was funded by the EC under IST-2001-32243.

References

1. I. Foster and C. Kesselman (eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999.
2. The CrossGrid Project: `http://www.crossgrid.org`
3. Message Passing Interface Forum, MPI: A Message Passing Interface Standard, June 1995, `http://www.mpi-forum.org`
4. Gomes, J. Marco, et al., "First Prototype of the CrossGrid Testbed", Proc. AcrossGrids Conf., Santiago, February 2003; ISBN 3-540-21048-2 Lecture Notes in Computer Science 2970: 67-77, 2004.
5. The DataGrid Project: `http://www.edg.org`
6. The LHC Computing Grid Project: `http://lcg.web.cern.ch`
7. The Globus Project: `http://www.globus.org`
8. The Local Configuration software: `http://www.lcfg.org`
9. The CrossGrid Developers' Guide: `http://gridportal.fzk.de/websites/crossgrid/iteam/devguide/devguide-html`
10. Marian Bubak et al., CrossGrid deliverable D5.2.3, Full Description of the CrossGrid Standard Operational Procedures and Specification of the Structure of Deliverables, 2002
11. The Majordomo software: `http://www.greatcircle.com/majordomo`
12. The CrossGrid central software repository and development server, GridPortal: `http://gridportal.fzk.de`
13. The Savannah software: `http://gna.org/projects/savane`
14. The Virtual Rooms VideoConferencing System: `http://www.vrvs.org`
15. CVS Documentation: `http://www.cvshome.org`
16. Autobuild entry page: `http://savannah.fzk.de/autobuild/i386-rh7.3-gcc3.2.2`
17. Autobuilt RPMs: `http://savannah.fzk.de/distribution/crossgrid/autobuilt`
18. cg-lcfg-go: `http://cvs.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp4/config/cg-wp4-lcfg-server-files/cg-lcfg-go`
19. Jorge Gomes et al., "CrossGrid Deliverable D4.1 Appendix D: `Middleware Test Procedure`"
20. CVS Documentation: `http://www.loria.fr/~molli/cvs/doc/cvstoc.html`

DNS-Based Discovery System in Service Oriented Programming

Maurizio Giordano

Istituto di Cibernetica "E. Caianiello" - C.N.R.
m.giordano@cib.na.cnr.it
http://www.ais.cib.na.cnr.it/m.giordano

Abstract. *Service Oriented Programming* (SOP) is an emerging paradigm for grid computing and distributed applications development. SOP models an application as a composition of *services*, i.e. software components either local or remotely provided by third-party organizations. In SOP service information, like location and interface description, is often stored in registries (like UDDI) distributed in internet. Before using a service, applications need to know in advance its endpoint address or the location of the registry where to look for service information. In this scenario, dynamic discovery of service registry as well as inter-registry co-operation would be a very desirable feature. This paper proposes an approach for service discovery based on *DNS messaging*. It provides applications with DNS-based querying mechanisms to publish and discover on the network either the location of service registries or the endpoints of web services. This approach was implemented in a middleware system that uses the *Multicast DNS* technology and the *DNS-based Service Discovery* specification to provide respectively the communication infrastructure and a standard naming convention for service registries and web services advertisement and retrieving in a LAN.

1 Introduction

The growth of internet and the emerging e-business have imposed a service oriented approach in the development of distributed applications. Companies are realizing that they can achieve significant cost savings by outsourcing parts of their IT infrastructure to outside service providers. Indeed decentralization allows potentially the access to a huge amount of resources available in internet.

IT infrastructure decomposition into services occurs also inside the enterprise. Frequently the enterprise infrastructure grows up integrating different applications developed in incompatible programming models and software systems. In other cases, the enterprise forces the infrastructure decomposition to privilege the insulation of application modules from the underlying computing platform.

In grid computing [11] a set of resources that are not subject to centralized control can be combined to provide services to users that need them. The idea of making available hardware and software resources through the network in

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 840–850, 2005.

large-scale computing environments raises problems of resource discovery, resource interoperability, security. The new service oriented paradigm seems to be a promising approach for the Grid to address these problems, supported by the development of new industrial technologies.

Therefore *Service Oriented Programming* (SOP) [4, 5] is an emerging programming model to develop distributed applications for enterprise IT infrastructures and computational grids. SOP models applications in terms of *services*, i.e. software components that communicate via message exchange using the web as a "transport" *medium*. In this scenario, *Web Services* [18] is an emerging technology for the development of large-scale service oriented distributed systems.

Service oriented applications use services that can be either local or provided by third-party organizations. Service providers are software infrastructures, for instance application servers, exposing on the network software components (services) to be used by external applications. Service information is often stored in registries (UDDI [30] and ebXML [10]) distributed in internet. Registries store and publish several data describing services, like location, access methods and communication protocol. They are also used to classify services in taxonomies.

Before binding and invoking a service, applications need to know in advance either the location, the interface description and the binding protocols for that service, or the location of registries maintaining and publishing this information. In the second case applications can query registries and gather the required service information. In this context, dynamic discovery of service registry as well as inter-registry co-operation would be a very desirable feature.

In this work we propose an approach to provide applications with mechanisms based on *DNS messaging* for service publishing and discovery on the network. The main advantage of this approach is the use of existing, widely accepted and consolidated DNS technologies and APIs to accomplish service advertisement and dynamic discovery. In this scenario applications use DNS to discover in internet services location and interface constraints. Moreover applications can query DNS servers to be notified of the presence of service registries in the network, and then query and retrieve service information from them.

Service information is stored on DNS servers according to the *DNS-based Service Discovery* (DNS-SD) [6] specification, that is a way to use DNS records and APIs to store and query information beyond host address-to-name translations, in particular information on hardware and software systems (services).

In this work we present an implementation of this approach based on the *Multicast DNS* (mDNS) [7] technology and the DNS-SD specification, which provide respectively the communication infrastructure and a naming convention for services advertisement and retrieving on the network. The implementation is restricted to a LAN. Nevertheless the service discovery approach here presented has not this limitation, since it can also be adopted in unicast DNS *client-server* systems using *Dynamic DNS-update* [31] features for service information storing.

The rest of the paper is so organized: section 2 introduces the service discovery problem in service oriented programming presenting motivations for this work; section 3 describes the architectural approach and the implementation of our

system; section 4 makes a comparison of our approach with some related works; the same section reports some concluding remarks.

2 Service Discovery

Web Services[1] [18] is an emerging technology that differs from other approaches, like CORBA [22] and Java RMI [26], in focusing on simple and internet-based standards to address heterogeneous and distributed computing. Web Services is a *Service Oriented Programming* technology for the design of application as composed by a core system and software services linked and used *on demand*.

The big software companies are investing a lot of efforts and money to push SOP as the new programming model for web-based distributed applications. In fact, many vendors are selling software products, like application servers[2] and web application development frameworks with built-in functionalities to support web services development and life-cycle, from service publishing/providing (server-side), to service binding/execution (client-side). Examples of these frameworks are Microsoft .NET [21], SUN J2EE [24], and IBM WebSphere [16].

Web Services working groups [18] defined several XML-based standards for service interface definition (WSDL [8]), discovery (UDDI [30]) and binding protocols (SOAP [13]). These specifications do not provide technical solutions to some open issues, like service advertisement and dynamic discovery on the network. Before using a web service, applications need to know the service location, interface constraints and protocol; they get these data by processing the service WSDL file previously localized in internet. Since WSDL files are often stored and published by registries (UDDI and ebXML [10]) distributed in internet, applications can find services only on registries whose location is known in advance.

A desirable feature of service oriented applications is to make the interaction with external services independent from their location and implementation. One approach is the *naming-service* facility, i.e. a mapping between an application-level naming convention and the physical location and implementation of services. An example is the *Java Naming and Directory Interface* technology [25], that provides a set of naming/directory services to make Java applications transparently access resources (databases and file servers). A limit of this approach is the application dependency on Java programming language and environment.

Another approach to service dynamic resolving and binding on the network is *service brokering* [17]: proxy objects (*brokers*) distributed on the network store service information and advertise it upon application requests. A typical problem with these solutions is the use of proprietary protocols. Other service brokering solutions, like JINI [27] and SLP [15], are not scalable to the requirements for massive brokerage of services in internet or they are tied up with some language.

[1] "Web Services" is the technology acronym; in the rest of the document we call "web service" any software component (service) developed with this technology.

[2] An "application server" is a software framework that provides a complete environment for web application development, maintenance and support.

Other approaches, like *e-speak* [12] (discussed in section 4), use distributed directory systems and related protocols, like LDAP [20], to build a virtual-shared distributed information space to guarantee to applications an unique service/resource naming domain dynamically reconfigurable and re-mappable on the specific entity locations.

3 The Proposed Architecture

We propose to use existing, widely accepted and consolidated DNS technologies and APIs to achieve service advertisement and dynamic discovery in internet. The advantage of relying on DNS messaging to accomplish these tasks is twofold:

1. *Portability* - Everyone in internet uses the DNS. There by applications do not need to be compliant with proprietary protocols and/or integrate specific software solutions to access the service information lookup and storing facilities: they simply need to communicate with DNS servers in internet through DNS-based messaging.
2. *Scalability* - This is by far the most important issue for service discovery in wide-area network distributed applications. Although at the moment our experimentation is limited to LAN environments, we expect a service discovery system based on the traditional unicast DNS *client-server* architecture could benefit from the DNS scalable behavior in WAN settings.

We developed a software system, named *DNS-based Service Discovery Architecture* (DNS-SDA), to provide DNS-based service registration and lookup facilities to applications. By using DNS-SDA, applications are enabled to dynamically gather information on services, in particular service location, interface ports and communication protocols, regardless of the service type. Before describing the system architecture we list here some assumptions and the design strategies:

- **UDDI-based service repository** - Service information, like interface, communication protocol and binding endpoints are described in WSDL [8] documents (files). The service description documents can be stored and classified in UDDI registries. UDDI [30] is the *de facto* standard specification for distributed web-based information registries of web services. In our architecture service information, in the form of WSDL documents, can be either retrieved by querying UDDI repositories or localized directly on the network by means of the WSDL file URLs. The service provider can be an application container (like the Apache Axis framework [3]) that exposes (and maps) under certain URIs the service software components.
- **DNS-based service discovery** - DNS records are used to store either UDDI registry information, like UDDI server address and protocol, or web services information, like WSDL document URL. Once created, DNS records are registered as entries on DNS server databases, and can be queried using DNS APIs (available in several programming languages) and packet formats.

With this approach any application can interact with DNS servers to perform service advertisement and discovery. We adopt the DNS-SD [6] naming convention to enumerate instances of service registries or web services.

– **Plug-in & web service implementation** - We designed DNS-SDA as a self-contained software component with methods implementing service information registration and querying. The DNS-SDA software component communicates with applications via DNS messaging and can be integrated into applications in a multi-tier architecture design. We have also wrapped up the component into a web service, providing a WSDL description of the DNS querying functions. Applications supporting Web Services protocols, independently from the underlying computing platform, may access the service advertisement/discovery system using the DNS-SDA web service *on demand*.

Services and UDDI registries are enumerated and advertised on the network with names following the *DNS-based Service Discovery* (DNS-SD) [6] convention and defined in subsection 3.1. This naming is composed by a prefix, i.e. the unique name of the repository (or service) instance, and a dot-separated suffix that groups repository (or service) instances according to the internet domain and supported protocols. Names and associated resource descriptions (like, IP address, port number, and *key-value pairs*) are stored in form of *DNS resource records* (DNS-RR) [14] that can be queried and managed through DNS messaging. DNS message exchange may rely on a traditional DNS *client-server* architecture in a WAN, or on a DNS *sender-responder* symmetric infrastructure based on the *Multicast DNS* (mDNS) [7] technology which is limited to a LAN. More details on the mDNS architecture are reported in subsection 3.2.

The current DNS-SDA implementation (see figure 1) uses the mDNS technology as communication infrastructure: each host in a LAN runs its own *DNS responder* (server) listening on a host port and communicating over the multicast UDP/IP address. The system implementation is configured as a decentralized and symmetric DNS system limited to a LAN, where each host may play the role of both *DNS responder* and *DNS sender*.

Our service discovery approach can be implemented using traditional unicast DNS *client-server* infrastructure working in WAN environments: remote storing of service information on DNS servers, external to the local LAN, can be achieved via DNS special messages, namely *Dynamic DNS-update* [31] requests. Information about services and UDDIs can be packed in DNS-SD records and remotely saved on DNS servers in internet. However, the majority of DNS server deployments restrict (for obvious security reasons) the ability to update DNS records completely or to only a few known hosts.

In figure 1 we present an application of DNS-SDA, with a configuration of two application servers and one client: each application server runs on a different host: the topmost one hosts a UDDI application, for instance the opensource JUDDI [1] software running in the Apache Tomcat container, while the application server at the bottom is a web services container, for instance the Apache Axis framework [3]. In this system configuration service information, in the form

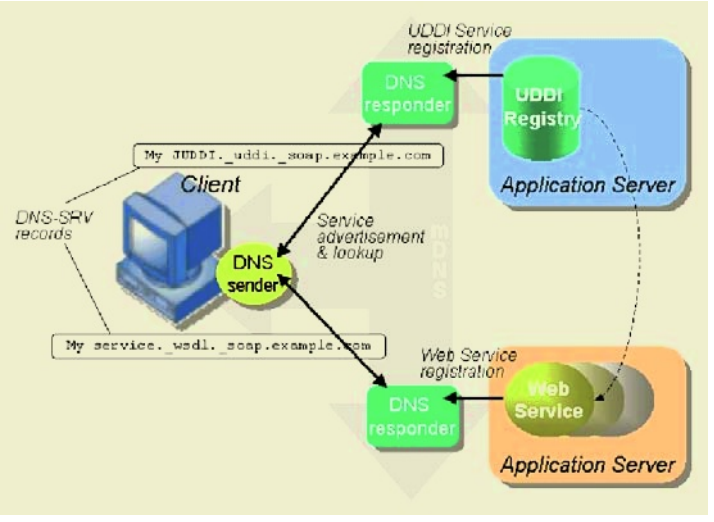


Fig. 1. The DNS-based Service Discovery Architecture

of WSDL files, may be stored in UDDI registries, or it can be accessed through the service WSDL URLs provided and published by the web service container.

3.1 DNS-Based Service Discovery

DNS-based Service Discovery (DNS-SD [6]) is a naming convention to enumerate and classify available software and hardware services (and resources), like web servers, ftp servers and UDDI servers, in the network using special *DNS resource records* [19], namely *DNS SRV records* [14]. While DNS-SD specification is a draft on the IETF standards track, there are several DNS-based implementations adopting this specification for network resource discovery [29, 23].

Service instances are grouped by *type* and *domain*. This information together with service instance *name* and optional *key-value pairs* are all stored in DNS SRV records. Any application looking for a service type in a searching domain can use standard DNS messaging to query DNS SRV records and discover the list of service instances of that type and belonging to the specified domain.

A service *type* is a dot-separated pair of names with format `<prot1>.<prot2>`, normally interpreted as the service protocols (in the protocol stack order). For example, by querying DNS SRV records with the name `_http._tcp.example.com.` applications discover all service instances using HTTP over TCP protocol in the `example.com` domain (i.e. web servers). The assumption is that a service name identifies several instances of the same service, i.e. same interface and capability.

In our approach we used this naming convention to enumerate and advertise on the network UDDI registry servers, i.e. repositories for the classification, publishing and storing of web services information. Those registries can be queried for service discovery in the UDDI API specification adopting the SOAP [13]

protocol for message exchanging. To denote a set of UDDI registries in our <domain> we used the name _uddi._soap.<domain>. A client could query the DNS for PTR records of _uddi._soap.<domain> and would be returned a list of named instances of UDDI servers. In the example of figure 1 the messaging is:

```
PTR:_uddi._soap.example.com -> SRV:My JUDDI._uddi._soap.example.com
                            -> SRV:My JWSD._uddi._soap.example.com
```

A client application could then display the list of discovered UDDI servers. Assuming the client wants to use the "My JUDDI" registry [1], it can resolve that DNS SRV record associated to the UDDI registry instance, thus retrieving the UDDI server URI from the record description field:

```
SRV:My JUDDI._uddi._soap.example.com -> http://myhost.example.com/juddi/
```

The client application uses the http://myhost.example.com/juddi/ URI to contact the UDDI server; then it starts querying the registry exchanging messages in the UDDI API packet format and using the SOAP transport protocol.

We used DNS-SD naming convention also to enumerate and advertise on the network web service descriptor files (WDSL). To denote them we used the name _wsdl._soap.<domain>. A client could query the DNS for PTR records of _wsdl._soap.<domain> and would be returned a list of named WSDL files. An example of DNS message exchange for web services discovery could be:

```
PTR:_wsdl._soap.example.com -> SRV:My Service._wsdl._soap.example.com
SRV:My Service._wsdl._soap.example.com ->
                            http://myhost.example.com/axis/MyService.wsdl
```

3.2 mDNS

Multicast DNS (mDNS) [7] is a joint work of the IETF Zero Configuration Networking [32] and DNS Extensions [9] working groups. The idea behind mDNS is to allow a group of hosts on a LAN, without using conventional unicast DNS servers, to co-operatively manage a collection of DNS resource records allowing client applications on the same LAN to query those records. As we have seen before, DNS SRV records can be used to keep information on general services provided either by hardware devices and software components on the network.

mDNS reserves the DNS namespace, named .local., for use by clients in the LAN. A client application connects to the mDNS multicast IPv4 address, then sends queries and receives responses to port 53 via DNS-based messaging.

mDNS architecture is based on *senders* and *responders*. A *sender* is any software sending a query to the mDNS multicast address. A *responder* is any software that listens to (but not necessarily responds to) a query on the same address. The mDNS architecture is symmetric since any responder may also be a sender, while a software not configured as responder cannot be a sender.

A sender sends to the multicast address DNS queries for any type of resource record (e.g. A, PTR, SRV, etc.) in the .local. domain. A responder must answer

a multicast query only for the name for which it is authoritative. Differently from traditional DNS, a responder is not authoritative for subdomains of the domain of competence.

With reference to the first example of DNS messaging in the previous subsection, if a responder is authoritative for the domain name `_uddi._soap.local.`, when it receives a multicast DNS query for a PTR record it responds with a pair of SRV records giving information about two UDDI registries. The same records can be registered in a different domain name, named `_uddiv2.0._soap.local.`, for which it is authoritative another mDNS responder on the same LAN.

In our approach DNS SRV records keep information about web services or UDDIs and are managed in a mDNS environment, i.e. mDNS is the communication infrastructure for DNS-SDA. The main limitation of mDNS is its applicability to local-area network. Moreover we expect that new applications in the future will use multicast DNS as a communication infrastructure, thus producing and amount of traffic beyond the mDNS conventional one. DNS-SD is not limited to mDNS since DNS-SD is orthogonal to mDNS; the DNS-SD convention can also be adopted with traditional unicast DNS *client-server* infrastructure in a WAN.

4 Related Works and Concluding Remarks

Today UDDI [30] is the leading technology for web service information registries for storing, classification and retrieval of services in order to support the design and the interoperability of several software applications in grid–like distributed computing and enterprise IT infrastructure development.

The main architectural change in the UDDI version 3.0 specification is the concept of "UDDI federation" for registry interaction, i.e. the facility to group registries in an *affiliation* by choosing appropriate policies. The main purpose of a UDDI federation is twofold: 1) to guarantee a global namespace for the management of UDDI entity keys in a hierarchical organization of registries: 2) to support controlled redundancy of core data structures among registries, in order to make consistent multiple copies of the same service information stored in a distributed global registry.

While UDDI technically allows the interoperability of registries, it does not directly support it: in fact UDDI specification leaves policies and implementations of the registry interaction to the registry operators. Also with UDDI federated in groups, mechanisms and strategies to propagate registry querying throughout the UDDI federation have to be designed and implemented in the application or in middleware systems offering service discovery support.

In service oriented programming model the availability of dynamic discovery mechanisms for service registry location is a very desirable feature. In fact, before using a remote service, applications have to find that service information in either a centralized public registry (URB) or in some private registries distributed in internet whose location has to be known in advance.

In grid environments, like Globus Toolkit v.3 [28], services are discovered by querying an *Index Service*. The Index Service maintains a registry of local

grid services. the *Service Data Aggregation* mechanism of GTK3 allows to implement an inter-registry co-operation mechanism: service data coming from an Index Service in a GTK container can be subscribed and notified to Index Services in other containers. Although this mechanism can be implemented using the GTK Index Services capabilities, the main limitation of this approach is the static configuration of inter-registry communication topology among OGSA islands.

In our approach service registry owners and service providers can use DNS infrastructure and messaging to enter the community announcing their presence in the network, the availability of offered services and their descriptions. This can be achieved in a way similar to how we register in the DNS a new computer for hostname-to-address resolution. Client applications get a service entry without knowing in advance the location of the owning registry. Once identified a set of registries, a client can query them for service retrieving. In this scenario the service search space is enlarged beyond the private registry space.

A similar approach is adopted by *e-speak* [12] from HP. E-speak consists of *Service Framework Specification*, *e-services Village* and the *e-speak Service Engine*. E-speak goal is to implement cross-compatibility between different IT infrastructures sharing the use of e-speak services. E-speak service discovery operations are implemented in a service separated out of the core functionality. *E-speak Service Engine* can take a service query and search all known repositories for required service descriptions. The e-speak mechanisms for service registration, notification and discovery are based on LDAP [20]. E-speak allows to group different e-speak engines. Different groups in a network use LDAP to share the service information (groups are combined in an *abstract community*).

A limitation of e-speak is that it does not have protocols using standard technologies for the maintenance of its groups. The match making specification and the interfaces for e-speak is planned to be made UDDI compliant by HP. Another limitation of e-speak is that every software application joining the e-speak community (*e-services Village*) have to support the e-speak platform.

Our approach shares some aspects of the e-speak architecture in designing the lookup engine as a separate service from the core functionalities. In fact in our system the service registry or provider lookup facility is implemented as a new component that can be either included in the application as a middleware layer that provides a defined API, or wrapped up to build a web service exposing methods for service discovery to be bound and invoked by external applications.

Our approach differs from e-speak in the adopted protocols for service advertisement and lookup. E-speak uses LDAP as low-level protocol for service information and grouping system, while it defines a proprietary protocol that applications need to support to interact with the e-speak service system.

In our approach we use the DNS as high-level protocol for service information publishing and retrieving. Since the DNS is widely used in internet, there is no need to be compliant with proprietary protocols to access the service information

lookup and publishing facility. Software systems can publish their service information in DNS records stored on DNS servers and available for consultancy.At the same time, applications can query DNS records in the network for service entries discovery, like common web browsers do in resolving hostnames.

We offer to applications a DNS-based API for service registration and lookup. We think that service discovery facilities should be implemented as a separate software component to be plugged into applications in a multi-tier architecture design or used as an external service. Other related and critical services, like authentication and security, could be added to the system in a similar way.

References

1. Apache Web Service Project: JUDDI. http://ws.apache.org/juddi/
2. Apache Jakarta Project: Tomcat. http://jakarta.apache.org/tomcat
3. Apache Web Service Project: Axis. http://ws.apache.org/axis/
4. Bieber G., Carpenter J.: Introduction to Service-Oriented Programming. http://www.openwings.org/download.html. (2004)
5. Sillitti A., Vernazza T., Succi G.: Service Oriented Programming: A New Paradigm of Software Reuse. LNCS **2319**:269–280. (2002)
6. Cheshire S., Krochmal M.: DNS-Based Service Discovery (Internet Draft). Apple Computer, Inc. http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt. (2004)
7. Cheshire S., Krochmal M.: Multicast DNS (Internet Draft). Apple Computer, Inc. http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt. (2004)
8. Chinnici R., Gudgin M., Moreau J., Weerawarana S.: Web Services Description Language (Vers. 1.2 working draft). http://www.w3.org/TR/wsdl12. (2003)
9. DNS Extension Charter. http://ietf.org/html.charters/dnsext-charter.html
10. ebXML: Enabling A GlobalElctronic Market. http://www.ebxml.org/
11. Foster I. *et al.*: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Intern. J. Supercomputer Applications. **15**(3). (2001)
12. Graupner S. *et al.*: e-Speak an XML document interch. engine. LNCS **2215**. (2001)
13. Gudgin M., Hadley M., Mendelsohn N., Moreau J., Nielsen H.: SOAP Version 1.2 Part 1: Messaging Framework. http://www.w3.org/TR/soap12-part0/. (2003)
14. Gulbrandsen A. *et al.*: A DNS RR for specifying the location of services. Network Working Group RFC-2782. http://www.faqs.org/rfcs/rfc2782.html. (2000)
15. Guttman E.: Service Location Protocol: Automatic Discovery of IP Network Services. IEEE Internet Computing. **3**(4):71–80. (1999)
16. IBM WebSphere. http://www-306.ibm.com/software/info1/websphere/
17. Jagatheesan A., Helal S.: Sangam: Universal Interop Protocols for E-Service Brokering Communities using Private UDDI Nodes. IEEE symp. Computer and Communications - ISCC'03. (2003)
18. Kreger, H.: Web Services Conceptual Architecture (WSCA 1.0). http://www-4.ibm.com/software/solutions/webservices/. (2001)
19. Mockapetris. P.: Domain names - implementation and specification. Network Working Group RFC-1035. http://www.faqs.org/rfcs/rfc1035.html. (1997)
20. Lightweight Directory Access Protocol (v3). Network Working Group RFC-2251. http://www.ietf.org/rfc/rfc2252.txt. (1997)
21. Microsoft .NET. http://www.microsoft.com/net

22. Vinoski, S.: CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. IEEE Communications Magazine. (1997)
23. Porchdog software. Howl. http://www.porchdogsoft.com/products/howl/. (2004)
24. Sun Microsystems: Java 2 Platform, Enterprise Edition. http://java.sun.com/j2ee/
25. Sun Microsystems: Java Naming and Directory Interface. http://java.sun.com/products/jndi
26. Sun Microsystems: Java RMI. http://java.sun.com/products/rmi
27. Sun Microsystems: JINI. http://www.jini.org
28. The Globus Toolkit. http://www-unix.globus.org/toolkit. (2004)
29. The mod_zeroconf site. http://www.temme.net/sander/mod_zeroconf/. (2004)
30. UDDI.org, UDDI Version 3.0 Published Specification. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm. (2002)
31. Vixie P. *et al.*: Dynamic Updates in the Domain Name System. Network Working Group RFC-2136. http://www.ietf.org/rfc/rfc2136.txt. (1997)
32. Zero Configuration Networking. http://www.zeroconf.org/

Experiences with Deploying Legacy Code Applications as Grid Services Using GEMLCA[1,2]

A. Goyeneche[1], T. Kiss[1], G. Terstyanszky[1], G. Kecskemeti[1], T. Delaitre[1], P. Kacsuk[2], and S.C. Winter[1]

[1] Centre for Parallel Computing, Cavendish School of Computer Science, University of Westminster,115 New Cavendish Street, London, W1W 6UW
[2] MTA SZTAKI Lab. of Parallel and Distributed Systems, H-1518 Budapest, P.O. Box 63, Hungary
gemlca-discuss@cpc.wmin.ac.uk
www.cpc.wmin.ac.uk/gemlca/

Abstract. One of the biggest obstacles in the wide-spread industrial take-up of Grid technology is the existence of a large amount of legacy code programs that is not accessible as Grid Services. On top of that, Grid technology challenges the user in order to intuitively interconnect and utilize resources in a friendly environment. This paper describes how legacy code applications were transformed into Grid Services using GEMLCA providing a user-friendly high-level Grid environment for deployment, and running them through the P-GRADE Grid portal. GEMLCA enables the use of legacy code programs as Grid services without modifying the original code. Using the P-GRADE Grid portal with GEMLCA it is possible to deploy legacy code applications as Grid services and use them in the creation and execution of complex workflows. This environment is tested by deploying and executing several legacy code applications on different sites of the UK e-Science OGSA testbed.

1 Introduction

There are many efforts all over the world to provide new Grid middleware concepts for constructing large production Grids. As a result, the Grid community is in the phase of producing third generation Grid systems that are represented by the OGSA (Open Grid Services Architecture) [1] and WSRF (Web Services Resource Framework) [2] standards. On the other hand relatively little attention has been paid to how end-users can survive in the rapidly changing world of Grid generations. The primary goal of our research is to construct a high-level Grid application environment where the end-users can:

[1] The work presented in this paper was initially supported by EPSRC funded project (Grant No: GR/S77509/01): A proposal to evaluate OGSA/GT3 on a UK multisided testbed.
[2] The integration of GEMLCA, P+Grade and other Grid tools and projects is currently supported by "*CoreGrid*", network of excellence in "*Foundations, Software Infrastructures and Applications for large scale distributed, Grid and Peer-to-Peer Technologies*".

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 851 – 860, 2005.

- Deploy and use any legacy code as Grid Services.
- Use these Grid Services in order to easily and conveniently create complex Grid applications.

In an ideal Grid environment users would be able to access Grid Services through a high-level user-friendly Grid portal. More than that, users would not only be capable of using such services, they could dynamically create and deploy new services, and also construct complex Grid workflows in a convenient and efficient way. All these services can be either specifically designed Grid Services, or legacy code programs deployed as Grid Services.

Some of these objectives are achieved by an integrated high-level Grid execution environment [3] consisting of Grid Execution Management for Legacy Code Architecture (GEMLCA) and the P-GRADE Grid portal. GEMLCA enables legacy code programs written in any language (Fortran, C, Java, etc.) to be easily deployed as a Grid Service without any user effort. The integration of GEMLCA with the P-GRADE Grid portal results in an OGSA-based Grid portal. Using this integrated solution legacy codes can be deployed as Grid Services and accessed by authorized users from the portal. As a consequence, legacy codes (either sequential or parallel ones) can be used as workflow components to build complex Grid applications.

In the current paper the integrated GEMLCA & P-GRADE Grid portal is presented describing experiences deploying and running different legacy applications as Grid Services.

2 Deploying Legacy Code Applications on the Grid

Many currently available industrial and scientific applications were written well before Grid computing or service-oriented approaches appeared. The incorporation of these legacy code programs into service-oriented Grid architectures with the smallest possible effort is a crucial point in the widespread industrial take-up of Grid technology.

There are several research efforts aiming at automating the transformation of legacy code into a Grid Service. Most of these solutions are based on transforming legacy code applications into Web Services outlined in [4] using Java wrapping in order to generate stubs automatically. One example is presented in [5], where the authors describe a semi-automatic conversion of programs written in C into Java using Java Native Interface (JNI). After wrapping the native C application with the Java-C Automatic Wrapper (JACAW), MEdiation of Data and Legacy Code Interface tool (MEDLI) is used for data mapping in order to make the code available as part of a Grid workflow.

Different non-wrapping approaches are presented in [6] and [7] but these solutions only define the principles of legacy code transformation, and do not specify an environment or tool to do the automatic conversion.

GEMLCA is based on a different principle. Instead of wrapping the application GEMLCA hides it behind a set of Grid Services leaving the legacy code untouched. The Grid Services layer communicates with the client in order to submit service requests, manages input and output parameters, and contacts a local job manager through Globus MMJFS (Master Managed Job Factory Service) to submit the computational jobs. To deploy a legacy application as a Grid Service there is no need for the

program source code and not even for the C header files as in case of JACAW. The user only has to provide the program attributes and parameters. The legacy code can be written in any programming language and can be not only a sequential but also a parallel MPI or PVM code that uses a job manager like Condor, and where wrapping can be difficult.

3 GEMLCA

GEMLCA is a Grid architecture with the main aim of exposing legacy code programs as Grid Services without reengineering the original code, and offering a user-friendly interface.

GEMLCA has been designed [8] as a three-layer architecture: the first (front-end) layer, offers a set of Grid Service interfaces that any authorized Grid client can use in order to contact, run, and get the status and any result back from the legacy code. This layer hides the second (core) layer, which deals with each legacy code environment and their instances as Grid legacy code processes and jobs. The final (back-end) layer is related to the Grid middleware where the architecture is being deployed. The present implementation is based on GT3 and is currently migrated to GT4.

GEMLCA conceptual architecture is shown in Figure 1 where we can identified a Grid client, a GEMLCA Resource which is composed of a set of Grid Services that provides a number of Grid interfaces in order to control the life-cycle of the legacy code execution and a Grid Host Environment provided by the deployment of Globus Toolkit 3 (GT3).

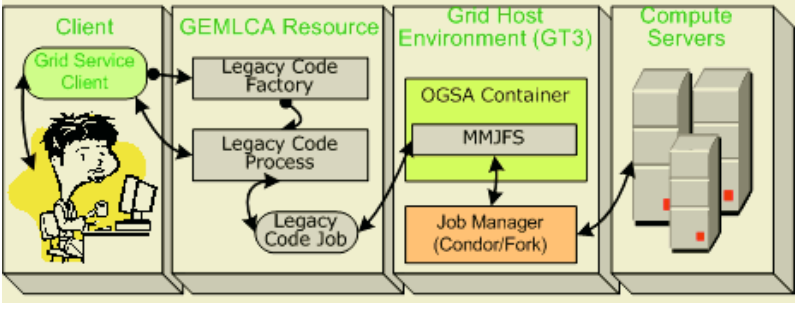


Fig. 1. GEMLCA functional architecture

In order to access a legacy code program, the user invokes the GEMLCA Grid Service client which creates a legacy code instance with the help of the legacy code factory. Following this, the GEMLCA Resource submits the job to the compute server through GT3 MMJFS using a particular job manager.

GEMLCA is been designed using three layers that encapsulate and divide the front end and core functionality with the Grid Host Environment dependant layer. At this time, a GEMLCA version using a GT3 Grid Host Environment is available where a GT4 dependant version is been under development and testing.

4 The Integrated GEMLCA P-GRADE Portal

The P-GRADE Grid portal is a workflow-oriented portal which main goal is to enable users to manage the whole life-cycle of creating and executing complex applications in the Grid. This is achieved allowing users to edit, execute and monitor Grid workflows composed of various types of Workflow components (sequential programs, MPI, PVM).

Our goal, as it was mentioned in the Introduction, was to enable end-users to deploy legacy codes as Grid Services, and use them as components of workflows with the least possible effort. The integration of GEMLCA with the P-GRADE portal produces a high-level Grid toolkit environment that achieves this goal. The integration has been done through the creation and use of a number of Grid clients.

In order to create a GEMLCA component in a workflow, the portal user is able to select a GEMLCA Resource and the required legacy code program which has already been deployed on the resource. To achieve this, the Grid GEMLCA client, embedded in the portal, allows the selection of a GEMLCA Resource and a legacy code application from the list returned. This client also allows the change of input parameters and upload of input files.

Once the workflow is completed and saved, the workflow manager, Condor DAGman [9], is used in order to manage GEMLCA jobs. In case of GEMLCA, the DAGMan's PRE, POST, and job submission scripts were modified. The PRE-script has been changed in order to call a GEMLCA client that creates an instance of the legacy code process returning a Grid Service Handle (GSH). Such GSH is used by a further GEMLCA client for setting its parameters, uploading input files using GridFtp and finally submitting the GEMLCA job. The POST-script has also been changed in order to download and make output files available to the user, and destroy the GEMLCA jobs. Alternatively, the output files will be transferred into the next legacy code environment of the workflow if it is required.

Another GEMLCA client is used for checking the status of the legacy code process and jobs in order to let the user know the status of the workflow and each of its components.

5 Legacy Code Deployment with GEMLCA

5.1 Legacy Code Program Deployment Using the P-Grade Grid Portal

Most of the solutions to expose legacy code programs as Grid Services require access to the source code. In contrast, in GEMLCA the only significant effort to be done is to create a Legacy Code Interface Description (LCID) file in XML format. The LCID file consists of three sections: the first section – "*environment*" - contains the name of the legacy code and its binary file, job manager to be used (Condor and Fork are supported in the current version of GEMLCA), maximum number of jobs allowed to be submitted from a single legacy code process, and minimum and maximum number of processors to be used; the second section – "*description*" - describes the legacy code in simple text format; the third section – "*parameters*" - exposes the list of parameters defining for each of them its name, friendly name, type (input or output), order, status

(compulsory or optional), file, command line, and the regular expression to be used as input validation.

Some users, without basic GEMLCA and XML knowledge, may find difficult to learn how to manually create LCID files. To support them, the generation of LCIDs and the deployment of legacy code applications have been automated using a new Admin Grid Service in the GEMLCA Resource and a new Grid portal interface.

The Admin Grid Service can be accessed through the GEMLCA Administration Portlet which is integrated into the P-GRADE Grid portal, and offers a number of interfaces in order to create the legacy code program deployment environment and its LCID file.

The Portlet, based on the Document Type Definition file stored in the GEMLCA Resource provided by one of the Admin Grid Service interfaces, creates and presents a deployment Web form on the fly. If the legacy code description provided by the user is correct, the Admin Grid Service is used to create the legacy code deployment environment and its LCID file, and Grid FTP is used to upload the legacy code program and its input files. After this point the legacy code is published and available as a Grid Service.

The GEMLCA Administration Portlet hides the syntax and structure of the LCID file from users. As a result, users do not have to know LCID specific details. For example, they do not have to be familiar with possible modifications in legacy code description after new GEMLCA releases.

5.2 Internal Legacy Code Deployment Description

Besides the use of the P-Grade Grid portal, a legacy code program can be also deployed manually by the GEMLCA Resource administrator or any general user with access to the GEMLCA Recourse server.

The general user is allowed to deploy legacy code programs within the user's home folder. Any deployed program could only be used through GEMLCA by the Grid users which are locally mapped to the local general user. These legacy codes are considered "*private"* to a set of Grid users. On the other hand, the GEMLCA Resource administrator can also deploy a "*public*" legacy code program that is available to any Grid user mapped in that Grid host. This technique implements the GEMLCA authorization of legacy codes.

After deciding whether a legacy code will be available to either a set of users (*private*) or anyone (*public*) in a given GEMLCA Resource, a folder within the base deployment folder has to be created that gives a unique *private* or *public* identification. The legacy code can be uploaded into this folder from the portal, manually copied or linked from any place in the server without changing the program's internal folder structure. The input files can also be copied or linked into this folder, or made available using a full path folder. Finally, to make the legacy code available the LCID file has to be created.

6 Deployment Examples

In this section the deployment of three legacy codes are described: MadCity traffic simulator [10], GAMESS-UK [11] and MultiBayes [12]. These programs were devel-

oped by the University of Westminster, the Daresbury Laboratory, and the University of Reading, respectively. The objective of these deployments is to publish these legacy codes as Grid Services in order to be available for external Grid users. In each example, the challenge faced is presented together with the solution implemented.

At the end of the chapter a list of constraints that have to be considered before deploying a program as a GEMLCA service is included. The constraints are based on the experience of deploying the previously mentioned legacy codes.

6.1 MadCity Traffic Simulator

In this example, a workflow for analysing urban car traffic on road was created that consists of three different components: a Manhattan road network generator, a traffic simulator, called MadCity, and a traffic analyser.

The Manhattan road network generator creates MadCity compatible car networks that are used as inputs for the simulator. MadCity is a discrete time-based traffic simulator that simulates car traffic on a road network, and shows how individual vehicles behave on roads and at junctions. Finally, the traffic analyser compares the traffic density of several simulations of a given city and presents a graphical analysis.

The main objective of this case study was to analyze and test the following points:

- Use of several GEMLCA Resources in a single workflow.
- Execute several legacy codes as Grid Services in parallel in a single and multiple GEMLCA Resources.
- Schedule the workflow and synchronize the Grid Services executions.
- Transfer files from one Grid Service to another in the same GEMLCA Resource and between different GEMLCA Resources.
- Display intermediate and final results in the portal.

In order to meet these objectives, the workflow is configured to use five GEMLCA Resources each one deployed on the UK OGSA testbed sites, and one server where the P-GRADE portal is installed. The first GEMLCA Resource is at the University of Westminster (UK), and runs the Manhattan road network generator (Job0), two traffic simulator instances (Job3 and Job6) and the final traffic density analyzer (Job7). Four additional GEMLCA Resources are installed at the following sites: SZTAKI (Hungary), University of Portsmouth (UK), the CCLRC Daresbury Laboratory (UK), and University of Reading (UK). One instance of the simulator is executed on each of these sites, respectively Job1, Job2, Job4 and Job5 (see Figure 2).

The MadCity network and turn files are used as input to each traffic simulator instance. In order to analyse the different behaviour of these instances, each one was set with different initial number of cars per street junction, one of the input parameters of the program. The output file of each traffic simulation is used as input file to the traffic analyser.

The described workflow was successfully created and executed from the P-GRADE portal installed at the University of Westminster. The workflow execution graph is shown in Figure 3.

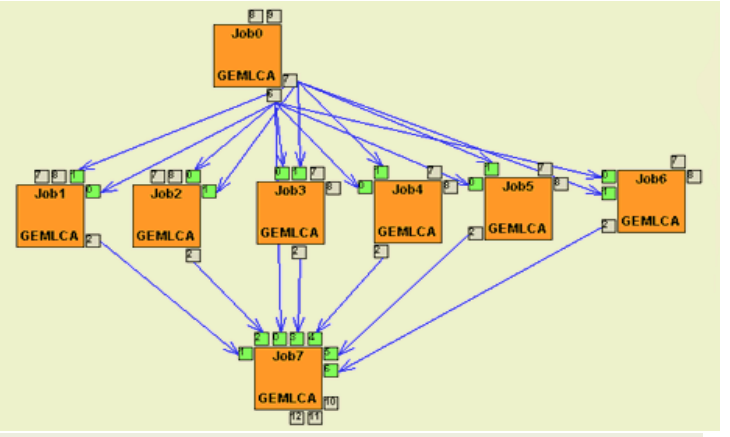


Fig. 2. Workflow graph for analysing road traffic

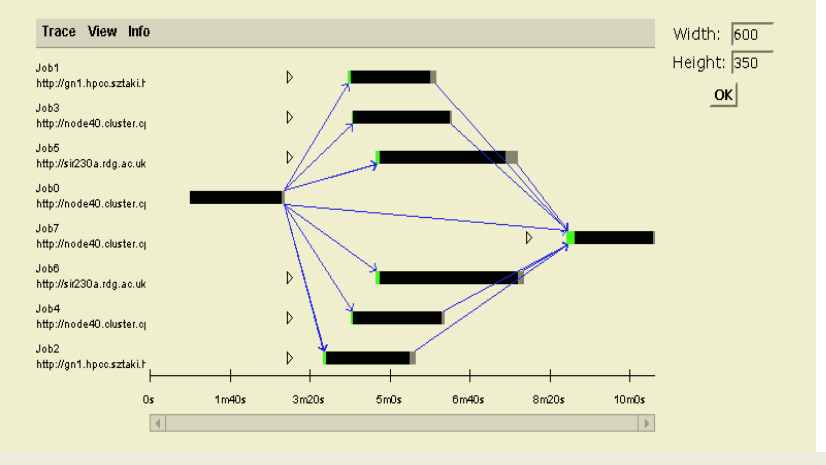


Fig. 3. Workflow execution graph

6.2 GAMESS-UK

GAMESS-UK is an ab initio molecular electronic structure program for performing SCF-, DFT-, and MCSCF-gradient calculations using a variety of techniques for post Hartree-Fock calculations.

The sequential version of GAMESS-UK, which has a complex folder structure with multiple files, was deployed. The application is an executable binary file that has one command line input parameter: a string that specifies the full path and name of the input file containing the program execution configuration. The ".in" file extension is internally added to this input file before reading it. The application is launched by a set of Bourne shell scripts that uses a number of ASCII text files to configure the application. Besides the standard and error outputs the legacy code generates two files where GEMLCA sends the final results.

The challenge of this deployment was to test GEMLCA with a complex structured legacy code and with run-time constrains, such as the input file implicit name in the input command line parameter, and the requirement of a full path definition in it.

The GAMESS-UK Grid Service was published using the $GEMLCAJOBPATH dynamic variable in order to set the input command line parameter, i.e. $GEMLCAJOBPATH/c2001_a. This variable is replaced at the time of running each job by the location of the volatile job environment that the job uses. The final input parameter defined in the LCID was set as a non-command line input file with the extension "in", i.e. c2001_a.in. Therefore, this file is included in each job environment but not listed as command-line parameter, given that another restriction of the legacy code is that it only accepts one and only one command line parameter.

The legacy code was successfully deployed using two GEMLCA Resources loaded at the University of Westminster and Daresbury Laboratory and tested from the P-Grade Gird portal running at University of Westminster.

6.3 MultiBayes

MultiBayes is an application developed at the School of Animal and Microbial Sciences at University of Reading and used in the Phylogenetic Tree Construction. It generates a Monte Carlo Markov Chain sample of trees from DNA sequences of genes common to a group of species.

The serial version of MultiBayes, which consists of an executable dynamically linked binary that accepts an input file as a parameter and creates three output files with the results, was deployed as GEMLCA Grid Service.

This legacy code was not as restrictive as GAMESS-UK but it presented a challenge to GEMLCA concerning the synchronisation between the P-Grade Grid portal and the GEMLCA Resources at the time of getting results due to its large output files. As a result, the P-Grade Grid portal workflow management post-script had to be tuned in order to cope with the Grid file transfer of results and their presentation.

The legacy code was successfully deployed at the University of Westminster as a GEMLCA Resource, and used from the P-Grade Grid portal also running at University of Westminster.

6.4 Legacy Code Deployment Restrictions

Testing GEMLCA by deploying legacy codes with different level of complexity and requirements led to a number of GEMLCA improvements that extended the list of programs to be deployed as GEMLCA Grid Services. The improvements were achieved modifying the GEMLCA core behaviour and adding new capabilities to the LCIDs.

Even after these improvements there are some constraints that have to be considered when selecting the legacy code to be deployed as GEMLCA Grid Service:

- The first and most important constraint is that the legacy code has to accept input parameters from the command line. Consequently, any legacy code with a user interface that accepts input data during the execution cannot be considered as a GEMLCA Grid Service.

- The legacy code has to write its results into a file or standard/error outputs in order to expose them to Grid users. Any other results stored in a different way, for example in external databases, could not be displayed. To avoid this problem a script should be attached to the legacy code program to write its output into local files.
- GEMLCA creates a volatile job environment where input files are uploaded before the legacy code is executed and output files are expected to be created. This environment is deleted when the Grid client destroys the job or when the Grid Service life-time expires. Another constraint to be considered is that the folder containing the input and output files have to be set dynamically. A legacy code with a pre-defined output file folder cannot be used because a Grid Service may be used by different users at the same time producing problems in the creation of output files.
- When an input parameter name is related to an output parameter, and a Grid client changes the input parameter name, it has to modify the output parameter name in order to let the portal know its new name.

7 Conclusions

In this paper the deployment of three legacy code applications as Grid Services using GEMLCA have been described. All deployments had different challenges that produced as a result, on top of the deployment of these programs, several improvements in GEMLCA. These changes enhanced the Grid architecture in order to guarantee its main objective, the deployment of legacy code without changing the program code, and also to increase the number of legacy codes programs to be accepted.

This exercise also produced a list of restriction that have to be considered when selecting a program to be deployed in GEMLCA. These restrictions are basically forced by the multi-user environment that has to be considered in Grid computing, and also the Grid object-oriented approach that does not make a full interactive program the best option for deployment.

Finally, with these examples, we demonstrated that the Grid environment composed of the integration of GEMLCA with the P-GRADE Grid portal enables Grid users to deploy legacy code applications, and to use them as Grid Services through a high-level user-friendly environment.

References

1. I. Foster, C. Kesselman, J. M. Nick, S. Tuecke. The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration. 2002. http://www.globus.org/research/papers/ogsa.pdf
2. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, S. Tuecke. From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Evolution Version 1.1 May, 2004, http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf

3. P. Kacsuk , A. Goyeneche , T. Delaitre , T. Kiss , Z. Farkas , T. Boczko: High-level Grid Application Environment to Use Legacy Codes as OGSA Grid Services, 5th IEEE/ACM International Workshop on Grid Computing, November 8, 2004, Pittsburgh, USA
4. D. Kuebler, W. Eibach: Adapting legacy applications as Web services, IBM Developer Works, http://www-106.ibm.com/developerworks/webservices/
5. Y. Huang *et al.*, "Wrapping Legacy Codes for Grid-Based Applications", Proceedings of the 17th International Parallel and Distributed Processing Symposium, workshop on Java for HPC), 22-26 April 2003, Nice, France. ISBN 0-7695-1926-1
6. T. Bodhuin, and M. Tortorella, "Using Grid Technologies for Web-enabling Legacy Systems", Proceedings of the Software Technology and Engineering Practice (STEP), Software Analysis and Maintenance: Practices, Tools, Interoperability workshop September 19-21, 2003, Amsterdam, The Netherlands.
7. B. Balis, M. Bubak, and M. Wegiel, "A Framework for Migration from Legacy Software to Grid Services", Cracow Grid Workshop '03, Cracow, Poland, December 2003.
8. T. Delaitre, A. Goyeneche, P. Kacsuk, T. Kiss, G.Z.Terstyanszky and S.C. Winter. GEMLCA: Grid Execution Management for Legacy Code Architecture Design., Conf. Proc. of the 30th EUROMICRO conference, Special Session on Advances in Web Computing, August, 2004, Rennes, France,
9. Condor DAGman, http://www.cs.wisc.edu/condor/dagman/
10. A Gougoulis, G. Terstyanszky, P Kacsuk, S C Winter: Creating Scalable Traffic Simulation on Clusters, PDP2004 Conf. Proceedings of the 12th EuroMicro Conference on Parallel and Distributed and Network-Based Processing, La Coruna, Spain 11-13th February, 2004.
11. GAMESS-UK, CCLRC, http://www.cse.clrc.ac.uk/qcg/gamess-uk/
12. MultiBayes, University of Reading, http://www.ams.rdg.ac.uk/

A Framework for Job Management in the NorduGrid ARC Middleware

Henrik Thostrup Jensen, Josva Kleist, and Jesper Ryge Leth

Danish Center for Grid Computing & Aalborg University, Denmark
{htj, kleist, leth}@cs.aau.dk

Abstract. This paper presents a framework for managing jobs in the NorduGrid ARC middleware. The system introduces a layer between the user and the grid, and acts as a proxy for the user. Jobs are continuously monitored and the system reacts to changes in their status, by invoking plug-ins to handle a certain job status. Unlike other job management systems, our is run on the client side, under the control of the user. This eliminates the need for the user to share a proxy credential, which is needed to control jobs. Furthermore the system can be extended by the user, as it is designed as a framework. This gives the users the possibility to adapt and extend it, to their needs.

1 Introduction

Computational grids, such as LCG2 [20], Grid3 [22], and NorduGrid ARC [9] are now finding everyday use, and grid technology is aiding research in areas such as physics, biology, nano technology, and computer science among others.

In large grids it is inevitable that resources from time to time fail, which may result in the failure of jobs, in which case the user will have to take action. Even without failures, the user might have to modify jobs, e.g., to move a job from one site to another with more free resources.

This poses new challenges for monitoring and reacting to state change of jobs, as the actions to perform depend on the type of job and cannot easily be generalized.

In this paper, we present the Job Manager[1] for managing jobs on NorduGrid ARC grids. The Job Manager runs on a machine trusted by the user, and is constructed such that the user can tailor the manager to fit specific needs for controlling jobs on a grid, by plugging in handlers for different situations. The system acts as a layer between the user and the grid, i.e., the users interaction with the grid happens through the Job Manager.

The paper is organized as follows: Section 2 presents work related to the Job Manager. Hereafter the NorduGrid project and ARC middleware is presented in Section 3; followed by reasoning for why the Job Manager is needed in 4. Section

[1] The Job Manager should not be mistaken for the Job Manager of Globus GRAM or in the Prospero Resource Manager.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 861–871, 2005.

5 describes the architecture of the Job Manager. Section 6 outlines possible uses for the Job Manager, and Section 7 concludes and presents future work.

2 Related Work

The concept of introducing a layer between the user and resource for managing jobs, has also been investigated by the Grid Service Provider [6] of the PROGRESS Project [5], and the EDG Resource Broker [17]. Both of these provides a job submission service to the grid for their users to submit jobs through.

The goal of the Grid Service Provider is to move functionality away from the client side, making grid user interfaces easier to develop and create. The typical user interface in PROGRESS is a web portal, and the Grid Service Provider provides a job submission service over a secure HTTP connection, making it easy for browsers to use.

The EDG resource broker is an integrated part of the EU datagrid architecture. It handles tasks related to job submission, such as file staging and brokering. Several resource brokers can co-exist and by creating a tuplespace between job submission services and the brokers a load balancing system has been created [17].

Dedicated job management on the grid is not a new concept either. The GEMS system [24], introduces a daemon to run locally on the resources, alongside the LRMS. This daemon is capable of monitoring hardware and software errors. In case of failure GEMS can restart the job, or migrate it to another site. To migrate a job, proxy credentials of the user are needed. These credentials are stored on a MyProxy server [19], and when submitting a job, the credential of the user is passed to GEMS. This architecture asserts that no resources will be compromised or have dishonest administrators for the credentials to be secure.

The Job Manager takes another approach by placing the system under the control of the user. This gives the user the possibility of modifying it to his or her own needs. Furthermore no proxy credentials has to be propagated to other sides, eliminating a potential security hazard.

3 NorduGrid

This section gives an outline of the NorduGrid project and the grid middleware it produces: The Advanced Resource Connector (ARC). The architecture and subsystems of ARC is described.

3.1 The NorduGrid Project

The NorduGrid project is a research and development project, launched in May 2001 with the goal of building a grid infrastructure suitable for production-level research tasks. The project develops and maintains a free grid middleware called the Advanced Resource Collector (ARC) [9], formerly known as the NorduGrid

middleware. Using this middleware, a testbed was started in may 2002 and has since august 2002 been in continuous operation and development. In November 2003 the testbed was considered stable and went to production level status. The grid it constitutes has continuously been growing and today consists of over 40 sites, providing over 5000 CPUs and 40 TB of data storage.

3.2 The ARC Middleware

The ARC middleware, here from referred to as ARC, is basically a distributed batch system, in which the user submits jobs directly to the resources. The user discovers resources by querying the ARC information system, where after each resource is queried for information necessary to do brokering which is also done on the client side. The user prepares a job description, which is uploaded to the resource found best suited, along with any local input files.

For a resource to be on an ARC grid it must run a set of daemons on a front-end machine. These daemons provides a set of grid services, and interfaces with the LRMS. An important design criteria in the ARC middleware is that software installation is only necessary on the front-end machine, no modification of the nodes should be necessary. This allows administrators to remain in control of their resources, while providing them to the grid. The rest of this section explains the different subsystems in ARC middleware.

ARC is build upon on the Globus Toolkit [2], however several components has been replaced or extended. The Grid Resource Allocation and Management (GRAM) [11] has been replaced by the grid-manager [15], which acts a gateway to the local LRMS system. The grid-manager supports the following LRMS: OpenPBS, PBSPro, Torque, Condor, Easy, SGE, and fork; the last being primarily for testing purposes. The grid-manager also handles staging of input and output files, i.e., it fetches input files needed by the job and uploads output files to storage elements after job completion. To authenticate to storage elements the grid-manager uses GSI proxy certificates [3], which the user supplies during job submission. Alternatively the user can fetch the output files from the cluster itself, since the grid-manager is run in combination with a GridFTP server [21].

The ARC GridFTP server [15] is not the standard provided from Globus. It does use the Globus GridFTP libraries though, but its structure has been rewritten to allow a plug-in structure, which allows the server to be used for job submission, where the grid-manager picks up the job submission files from a local job submission directory. A GACL plug-in [18] for fine grained access control lists is also provided. Finally a file plug-in provides a virtual directory, which does not have to reflect the layout on the file system.

The user interacts with an ARC grid, through a command line interface which supports submission of jobs, querying of the information system, copying of files etc. This interface is the primary way for a user to interact with the grid. Whereas grids such as EDG uses broker gateways [17], ARC users submits jobs directly to

resources; no third party is involved[2]. For this to work, the client needs access to brokering information and a local broker to find a suitable resource for the jobs to be submitted. This information is obtained by querying the ARC information system.

The ARC information system [16] is build upon the Globus MDS system [12], but does not use any of its caching abilities, and the LDAP schema provided by Globus been replaced with another suited to describe clusters. The information system is a multi-rooted tree, three levels deep. The two top layers consist of Grid Index Indices Servers (GIIS), and at lowest level the resources, i.e., cluster or storage elements. The resources contacts GIIS servers in the middle layer and provides them with their contact information. These GIIS servers again provides the top GIIS servers with their contact information (i.e., not the resources). The contact information propagation uses soft state registration, so if a resource falls out, it disappears from the grid after a timeout. The user can contact the top GIIS servers, where it can find contact information to the second layer GIIS servers. These again can be contacted for resource contact information, where after the resources can be queried for brokering information and a resource can be selected, by using a local broker.

Having selected a resource, a job description is prepared in xRSL[3] [23], an extension of RSL [1], which has been extended to support the architecture of ARC. The job description is submitted to the resource by uploading it to the GridFTP server of the resource, where the job plug-in will place the job description in a spool directory from which the grid-manager will pick it up. The GridFTP server will return a number to the user, which is appended to the URL of the GridFTP server, resulting in a string of the form:

```
gsiftp://grid.aau.dk:2811/jobs/1929610975219842091410212
```

This is known as a jobid, which is used as a future reference for the job, e.g., for cancellation or retrieval of output files.

4 Motivation

As mentioned in the previous, the user submits a job directly to a computing element. While this has worked reasonably well in ARC, it also means, that there is no one, except the user, to monitor jobs.

As long as the number of jobs stays small this is not a problem, but a user cannot manually handle thousands of jobs, which is not an uncommon amount of jobs to run on a grid. This means that there is a need for a system, which can monitor and control jobs on the grid. Such a system will be using the proxy credential of the user and should therefore be under the control of the user; running on a machine on which the user trust, so distribution of proxy credentials is minimized.

[2] This is not strictly true, since the grid-manager has various plug-ins, e.g., for accounting purposes, that may contact other sites, before starting the job.

[3] Should not be mistaken with XRSL, a resource specification language in XML.

Since users of the grid have different requirements and therefore want different actions taken for their jobs, it is important that a job management system is configurable, so users are able to define how jobs are handled. These requirements lean toward a framework where users can mix and match between existing ways of handling jobs and can plug in their own job "handlers" if necessary. Examples of such are resubmission of failed jobs, moving jobs that have been in queue to long or downloading job output files to the local workstation of the user.

It is clear that, such a system would automatize many trivial tasks for a grid user, which is not otherwise possible without providing a grid service with a proxy credential. Furthermore it gives the user the ability to create his or her own plug-ins for handling jobs thus providing a very flexible system. The next section describes such a job managing system, which we have developed for ARC.

5 Job Manager Architecture

This section describes the architecture of the Job Manager. However first, a small example of how the Job Manager works is given; also the choice of implementation language is argued for.

As mentioned in Section 1, the Job Manager introduces a layer between the user and the grid, meaning that when a user interacts with the grid, it happens through the Job Manager, like in the following example.

The user submits a job by invoking an RPC call to the the Job Manager containing the job description. The RPC call happens over a GSI connection to ensure privacy and that no other can interact with the Job Manager. The RPC server puts the job description in a database and marks the description, indicating that the job should be submitted. The status updater regularly iterates over all jobs in the database, updating their status, and handle them accordingly. It also discovers new jobs in the database and pass the job description to the handler module, which defines how to handle the job; the usual action for a new job being submission. This may seem as a lot of work to submit a job, but this scheme allow every action done to a job to be redefined.

The Job Manager is written in Python [13], a high level object oriented language. The reason for this was ease and speed of development compared to C and C++ the implementation languages of the Globus Toolkit and ARC. When using high level languages speed is often a concern, however the Job Manager is not compute intensive, but I/O bound, since it will usually be waiting for the network. Python is also highly portable and code written on one architecture usually runs on other platforms, with no or minimal porting effort.

Having given a brief example of how the Job Manager works and argued for implementation language, the internal architecture will be described. The architecture of the Job Manager is composed of four components: A GSI RPC server for client communication, a database for job descriptions and information, a status updater for refreshing the status of jobs, and finally a handler module which defines the actions that should be performed on the jobs. The following four sections describes each module.

5.1 RPC Server

The RPC server is the main access to the Job Manager. Through this, the user can submit jobs, cancel them, get the status of jobs, set policies, etc. The RPC server should only allow the user who started the Job Manager to connect. This is achieved by making the RPC call over a GSI connection which only allow a user the correct certificate and belonging private key to connect.

The Job Manager uses the SOAP RPC server provided by the pyGlobus project [7], since it uses GSI connections and allows per user authentication. Also SOAP [14] is a standardized and widely used protocol; however any RPC protocol which can run over a GSI connection could be used.

5.2 Job Database

The Job Manager uses a database for keeping job information. Using a database ensures persistence and consistency in the case of crash. Furthermore it reduces the complexity of writing the Job Manager and plug-ins, by removing the need for explicit locking of the job descriptions when reading and writing; instead transactions is used.

The Job Manager uses the Zope Object Database [10], since it integrates well into Python and is very lightweight compared to a full scale DBMS. The database is accessed transparently through a Python associative array, wherein all the keys and the mapped values are in the database.

With this scheme for accessing data, the natural way to store job descriptions is to give each job a key, which maps to its description. This key is created by making a hash of entropy collected from the system. The jobid, assigned to a job when submitted, cannot be used as key since a job can have several "instances", by being resubmitted, or moved, thus making it non persistent. The key is returned to the client, and must be used as identifier for the job between the Job Manager and client.

The database holds the following information about a job: Job description, job description type, and an associative array. Description is the job description received from the client and type is the type of the job description. Currently the only supported description type is xRSL. The remaining information about the job is kept in the associative array, which handlers can use for keeping information about the job. Using this scheme there is a minimal set of assumptions on what information the handlers will store, since they can store basically anything.

5.3 Status Updater

The task of the status updater is to update status information of jobs and invoke the handler module. The status information of jobs is pulled continuously from the information system at a regular interval, which can be defined by the user. If the retrieved information about the job is new it will be updated, and the handler will be invoked. If the status is not new, i.e., the job is in the same state as last checked, the handler will still be invoked. This is done since the Job Manager also reacts to no changes. This is useful, e.g., when a job has been

in queue for a long period of time, and should be moved. For this scheme to work, the time of job submission must be kept, and the handler compare the submission time to the current time, before any action is taken.

The algorithm of the status updater is as follows:

```
jobids = List()

foreach job in database
   if job is active
      foreach submission in job
         jobids.add(submission.jobid)

job_status = GetJobStatus(jobids)

foreach job in database
   foreach submission in job
      if submission.jobid in jobids
         if job_status[submission.jobid] != submission.status
            submission.status = job_status[jobid]
            new_status = true
         else
            new_status = false
         handler.invoke(submission, new_status)
```

First all the jobids of active jobs are collected. Only jobids from jobs which has one or more submissions are collected, so the status of finished jobs will not be retrieved. After collecting the jobids the status of them are pulled from the information system. Hereafter the submissions are once again iterated over and their status updated.

The reason for this scheme, is to have the transaction as short as possible when updating the status, and to make as few queries as possible, by only calling the `GetJobStatus` function once for each site, even if several jobs are given for it.

When using this algorithm, some states of the job may be skipped. E.g., the job may run for so short time that the first retrieved state indicate that the job has finished. This must be taken into account when writing handlers, however usually it will not have any effects, because states that can be skipped are usually the ones which the job should be in as short time as possible, i.e., queued or running. The states which we want to be sure to notice can usually not be skipped, since they are end states, e.g., finished or failed.

5.4 Handler Module

As mentioned earlier, the status updater regularly invokes the handler module. When invoked the task of the handler module is to see if any action should be done to the job, and if so, which. These decisions and actions are defined via plug-ins to the handler module, which we call handlers. The Job Manager supplies some common handlers, but all handlers can be redefined or omitted.

An important design criteria of the interface between the status updater and the handler module, is that the status updater does not know how jobs should be handled. Vice versa the handler module does not know how status information is retrieved or when it will be invoked.

It should be possible to mix and match handlers depending on the need of the user. Additionally, it should also be possible for the user to write handlers and plug them into the Job Manager.

Given these requirements, handlers should be separate and be able to plug into the Job Manager at runtime. Since Python already loads libraries at runtime, it has the functionality needed, and is also easy to use. For the ability to mix and match handlers we use mixins [8]. Using mixins, each handler is defined by a class, with a method being the code of the handler for a specific status. Additionally a top class, which provides the main handler interface and empty handlers (i.e., methods) for all the states are provided. On run time a new class is made by inheriting from the top class and different handler classes, possibly user provided. An instance of this class is then created. This object constitutes the handler. This scheme limits the system to only have one handler per job state. Having more than one handler per job state can quickly become confusing, so we do not regard this as a problem. If necessary the limitation can be overcome by having a handler inheriting from another, and reusing its functionality.

The handler exports one method, called `invoke`. It takes two arguments: The submission object, from which the status can be read, and a boolean indicating whether the status is new. When the method is called the job is queued in the handler object. This queue acts as a work queue, from which the handler can take work. The reason for this strategy is to allow the status updater to have its own thread and further isolate the status updater and the handler, by having a simple and small interface between them. There is a typical producer-consumer relationship between the two.

Due to the construction of the status updater, jobs usually arrive in batches. Handling all the jobs in parallel could easily overload the machine if the number of jobs is high, and handling them in sequence could possibly take a lot of time due to timeouts. Therefore the Job Manager uses a thread pool, and invokes each handler in a separate thread.

6 Uses for the Job Manager

This section presents possible uses for the Job Manager. We have already mentioned examples of such: Output file downloader, resubmission and job movement. The downloader retrieves the output files of a finished job to the local machine. This means that a scientist can submit jobs before going home, and have the output files locally available the next day. The resubmitter can resubmit a job in the case of failure, submitting the job to another site. Various failure reasons and how to react to them could be integrated in the handler. The job mover can move, i.e., cancel and submit a job, in the case that it has been

too long in queue. The handler could query the information system for more available resources, and only move in the case that a better site is found.

Besides the three examples, the Job Manager could cope with disappearing sites, by providing a handler for the jobs which status did not get retrieved. Since it is not uncommon for sites to be gone for a small period of time, such a handler should only submit the job again if the cluster has not responded for a longer period of time. Finally the Job Manager could be used as a part of a production system, e.g., the ATLAS Data Challenges [4], which requires running and monitoring several thousand jobs.

The Job Manager can also be used to build new user interfaces upon. Since the Job Manager provides a SOAP interface, instead of the multitude of protocols used by ARC, it should be more simple to do so. Such a construction also allows changes in the grid architecture, since the user can continue to use the SOAP interface. Finally multi-grid support could be build into the Job Manager, while still providing a single interface to the user, making a gateway for multiple grids.

7 Conclusion and Future Work

In this paper we have presented a framework for job monitoring. The system works by continuously monitoring jobs, and reacts to changes in their state by performing actions on the jobs. Since the Job Manager is run under the control of the user, it is possible for the user to define what actions should be done to jobs, by plugging in so called handlers. The system also eliminates the need for the sharing of proxy credentials since it is designed to run locally or on a machine that the user trusts.

The current status of the implementation is that the Job Manager is able to perform basic job manipulation tasks and react to status changes in jobs. Working on the implementation has revealed that the current ARC user interface is only suitable for the existing command line interface. In order to finish the Job Manager implementation, we are working on making a client library for ARC, making it possible to develop new application interfaces.

Future work on the Job Manager could be to make cooperative Job Managers, since they are currently a single point of failure, an undesirable feature, e.g., if the Job Manager would be used in a production system.

We would like to thank Niels Elgaard Larsen, Jakob Langaard Nielsen and Anders Wäänänen for providing us with valuable input, as well the people on the nordugrid-discuss mailing list for answering questions regarding the middleware.

References

1. The Globus Alliance. The globus resource specification language rsl v1.0. http://www-fp.globus.org/gram/rsl_spec1.html, May 2000.
2. The Globus Alliance. The globus toolkit. http://www-unix.globus.org/toolkit/, September 2004.

3. The Globus Alliance. Grid security infrastructure (gsi). http://www-unix.globus.org/security/, April 2004.
4. Nektarios Ch. Benekos and Armin Nairz. Welcome to the atlas data challenges. http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/DC/, June 2003.
5. Maciej Bogdanski, Michal Kosiedowski, Cezary Mazurek, and Malgorzata Wolniewicz. Progress – access environment to computational services performed by cluster of sun systems. http://progress.psnc.pl/English/ cgw02.pdf, December 2002.
6. Maciej Bogdanski, Michal Kosiedowski, Cezary Mazurek, and Malgorzata Wolniewicz. Grid service provider: How to improve flexibility of grid user interfaces? http://progress.psnc.pl/English/petersburg_ progress.pdf, June 2003.
7. Joshua Boverhof. Python globus (pyglobus). http://www-itg.lbl.gov/gtg/projects/pyGlobus, September 2004.
8. Gilad Bracha and William Cook. Mixin-based inheritance. In Norman Meyrowitz, editor, *Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications / Proceedings of the European Conference on Object-Oriented Programming*, pages 303–311, Ottawa, Canada, 1990. ACM Press.
9. The NorduGrid Collaboration. Nordugrid general information. http://www.nordugrid.org/about.html, September 2004.
10. Zope Corporation. Zope object database. http://zope.org/Products/ZODB3.2, June 2004.
11. Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. *A Resource Management Architecture for Metacomputing Systems. Lecture Notes in Computer Science*, 1459:62–??, 1998.
12. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 365–375. IEEE Computer Society Press, 1997.
13. The Python Software Foundation. What is python? http://python.org/doc/Summary.html, October 2004.
14. Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. Soap version 1.2 part 1: Messaging framework. http://www.w3.org/ TR/soap12-part1/, June 2003.
15. A. Konstantinov. The nordugrid grid manager and gridftp server - description and administrators manual. http://www.nordugrid.org/ documents/GM.pdf, July 2003.
16. Balazs Konya. The nordugrid information system. http://www.nordugrid.org/documents/ng-infosys.pdf, September 2002.
17. William Lee, Steve McCough, Steven Newhouse, and John Darlington. Load-balancing eu-datagrid resource broker. http://www.doc.ic.ac.uk/∼nfur/iceni/AHM2003/ edg.pdf, September 2003.
18. Andrew McNab. Gacl - a grid acl manipulation library. http://www.gridpp.ac.uk/authz/gacl/, November 2003.
19. J. Novotny, S. Tuecke, and V. Welch. An online credential repository for the grid: Myproxy, 2001.
20. LHC Computing Grid Project. Lhc computing grid project (lcg) homepage. http://lcg.web.cern.ch/lcg/, December 2003.

21. The Globus Project. Gridftp universal data transfer for the grid, September 2000.
22. The Grid2003 Project. Grid3. http:// www.ivdgl.org/grid2003/, October 2004.
23. O. Smirnova. Extended resource specification language. http:// www.nordugrid.org/documents/xrsl.pdf, October 2003.
24. S. Tadepalli, C. J. Ribbens, and S. Varadarajan. Gems: A job management system for fault tolerant grid computing. In *High Performance Computing Symposium 2004*, pages 59–66. J. Meyer (ed.), Soc. for Modeling and Simulation Internat., San Diego, CA, 2004.

Data Management in Flood Prediction

Ondrej Habala, Marek Ciglan, and Ladislav Hluchy

Institute of Informatics, Slovak Academy of Science,
Dubravska cesta 9,
845 07 Bratislava, Slovakia
{ondrej.habala, marek.ciglan, hluchy.ui}@savba.sk

Abstract. In this paper we present the data management tasks and tools used in a flood prediction application of the CROSSGRID [1] project. The application consists of a computational core - a cascade of three simulation stages, a workflow manager, two user interfaces and a data management suite. The project is based on the Grid technology, especially the Globus toolkit 2.4 and 3.2 and the EU DataGrid project. The paper begins with brief introduction to the application, its architecture and used technology. Most of the rest of the paper then describes in detail the various data management problems of the application and their solutions. The paper is concluded with a brief description of planned future work.

1 Introduction

Flood prediction became in recent years a serious problem (not only) throughout Europe. Floods have caused severe damage in most European countries and their prevention, or at least damage mitigation was seen as a very much desired research result. Therefore as one of the testing applications of the CROSSGRID[4] project was chosen a flood prediction application from Slovakia.

The rest of the paper briefly introduces the project CROSSGRID and the used Grid technology. Then there is an overview of the flood prediction application (FloodGrid), followed by the description of the constructed data management suite. The conclusion of the paper describes our plans for future work in this area. Grid computing emerged in recent years as a specialised track of distributed computing, with focus on large-scale distributed applications and massive, yet user-friendly resource sharing. The term Grid was coined in the 1990s and points toward the creation of a new network of interconnected resources, available on demand via the Internet - just as electricity is available through the power grid today. Vision of the Grid infrastructure was described[1] in 1999 and since then numerous scientific and some industrial projects based on the Grid technology emerged. Most of these projects develop the Grid middleware -

[1] This work is supported by EU 5FP CROSSGRID IST-2001-32243 RTD project and the Slovak Scientific Grant Agency within Research Project No. 2/3132/23.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 872–879, 2005.

a software layer implementing the vision and interfacing the various distributed resources to their users. Perhaps most known and most used is the middleware suite developed in the Globus Alliance[2], the Globus Toolkit[3]. This toolkit contains several modules, each implementing of its features - secure authorisation and authentication, job submission, information directory and data transfer. The contents of the toolkit went through serious development and its programming paradigm shifted over the years considerably. Yet we are concerned mainly with the second generation, which is the base of the CROSSGRID project and which uses custom protocols, based on UDDI, HTTP and FTP.

The project CROSSGRID started in 2002 and will end in the beginning of 2005. It is funded by the EU IST Directorate and developed by 21 partners from 11 European countries.

The main objective of CROSSGRID is to extend the Grid environment in Europe, especially for interactive applications, which did not receive enough attention in previous Grid software releases. The project is based on the second generation of the Globus Toolkit (Globus 2.4). It develops parts of middleware, as well as several application on which it can be tested. One of these applications, developed in the Task 1.2 of the Workpackage 1 of the project is the FloodGrid application.

2 FloodGrid Architecture

The grid prediction application of CROSSGRID, called FloodGrid aims to connect together several potential actors interested in flood prediction - data providers, infrastructure providers and users. The schema of a virtual organization supported by FloodGrid is illustrated in Fig. 1. The application has been in detail described before[5] and we will only state the most important facts about it, before focusing our view fully on the used data management suite.

The application's core is a cascade of several simulation models. At the beginning of the cascade is a meteorological prediction model (the ALADIN[7] and MM5[8] models are used), which receives its boundary conditions from outside of the cascade and computes temperature and precipitation predictions. These are then used in the second stage of the cascade - the hydrological model (HSPF[9]), which (based on precipitation data and watershed model) computes flow volume in selected points of the target river basin. This is then processed in the last stage of the cascade, in a hydraulic model - FESWMS[10]. This model uses actual terrain model to compute water flow in the area. Where the water hits area outside of the river basin, a flood is expected. All above mentioned outputs are visualized and available in nearly-interactive manner to the user.

The computational core is interfaced to the environment and to FloodGrid users by several other components. Most noticeable is the FloodGrid portal[6], which provides quite comfortable user interface (UI) to the whole application. Another UI, the Migrating Desktop is developed as a stand-alone Java application. The prediction cascade is controlled by a workflow service (based on more recent Globus 3.2) and the data management suite delivers the necessary data,

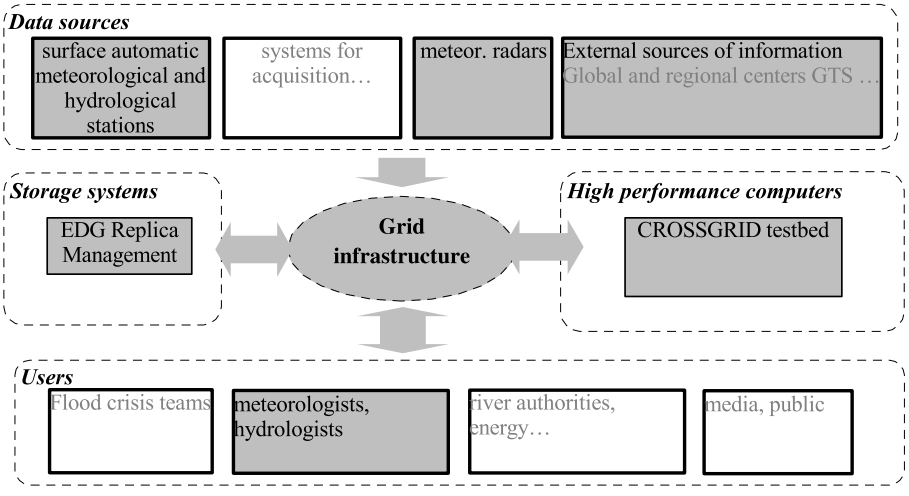


Fig. 1. Virtual organization for flood prediction

as well as catalogues, stores and replicates all computed results. This data management suite, its responsibilities and technical details are described in the rest of this paper.

3 Data Management

A problem solving environment of such a scale as is the FloodGrid application requires that its data be taken care of by a specialised software suite. This data management suite has:

- To enable delivery and storage of input data from sources outside of the PSE.
- To store all the computed (output) data, which has to be stored.
- To catalog all existing data and provide effective search facilities.
- To enable quick and straightforward access to all available data.

The data management software of FloodGrid is equipped with facilities that support all these goals. The rest of this chapter describes the facilities for transport of input data, for data cataloguing and lookup, for data storage and replication. The chapter ends with description of future development of data management software of FloodGrid.

3.1 Data Management Tasks in FloodGrid

General scheme of data management operations in FloodGrid is shown in Fig. 2. Only the most important operations are shown there.

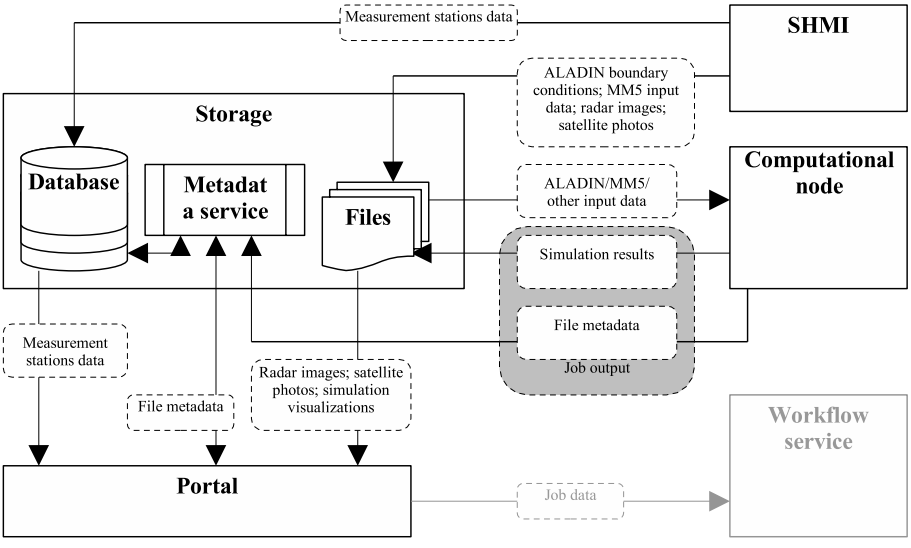


Fig. 2. Scheme of data management tasks in FloodGrid

As we can see, the Slovak Hydrometeorological Institute (SHMI) in Bratislava provides the input data of FloodGrid - radar and satellite images, measurement stations data and input data for both ALADIN and MM5 (input data of hydrological and hydraulic stages of flood prediction are computed inside the FloodGrid simulation cascade). Measurement stations data is stored in a RDBMS; all other data is just flat files. The measurement stations are graphically displayed in the FloodGrid portal, and also the metadata interface of FloodGrid portal requires transfer of metadata to/from the storage. While the measurement stations data ends its journey in the portal, the metadata is often used to find suitable data for a simulation job. Because of this connection between metadata and job definition document, also the workflow service and its connection to portal is displayed, although this part of the PSE is not incorporated in the data management suite described herein (therefore it is grey in Fig. 2).

The files with ALADIN and MM5 inputs are transferred (via a replica management software described later) to computational node when a job requires them. Files produced in the job are transferred back; their metadata descriptions are registered in the metadata service (also described later in this chapter).

3.2 Sources of Input Data

The input data of our flood prediction problem solving environment is divided into these groups:

- Radar images
- Satellite photos
- Measurement stations data

– ALADIN boundary conditions
– MM5 input data

All this data was, or is currently provided by SHMI. Transport of some of this data was implemented, deployed and tested for some time and then disabled because of SHMI Internet connection bandwidth considerations. Because all the transfers can be enabled anytime when needed, we will formulate our description in present tense to avoid confusion.

3.3 Transport of ALADIN Boundary Conditions and MM5 Input Data

The most important input data of our FloodGrid application is the ALADIN and MM5 input data, which is needed for the simulation cascade. All other simulated data is derived from one of these two sources. The data is computed at SHMI and transferred daily - uploaded to a computer inside the FloodVO. At this computer it is annotated with metadata and registerd with both replication suite and metadata service. Then it is ready to be used by a simulation job.

ALADIN and MM5 inputs are actually representation of the same physical values. MM5 inputs are currently computed at SHMI from ALADIN data (this may change in the future and MM5 input data may become independent from ALADIN input data; because of this the data is transferred separately, not computed in the FloodGrid application). Both ALADIN and MM5 are meteorological models, which can be used partly interchangeably. Anyway, we use them both because of their different internal implementation and abilities. Meteorological expert may find ALADIN more useful in some simulations, while in other the choice may be MM5.

3.4 Transport of Radar Images, Satellite Photos and Measurement Stations Data

This data can be considered 'complementary' to the ALADIN and MM5 input datasets. While it is not necessary for the simulation cascade, it may be very useful for hydrometeorological experts. The satellite and radar images provide both short-term weather information, on different scale and resolution. The measurement stations data is a source of real data, with which the computed ALADIN, MM5 and other inputs can be compared. If an expert wants to review a past situation, it is always better to have the real measured data than rely just on simulations. As was shown in Fig. 2, the measurement stations data are transferred to the portal, where they are available in the form of a graph.

3.5 Replica Management

The actual storage and maintenance of a coherent dataset collection is performed by a replica management software. It keeps track of the datasets, potentially stored at multiple places duplicitly (replicated). The creation of replicas of a single dataset may be well used for better security and protection against an

unwanted loss of the dataset because of a sudden storage device failure, as well as for better access to the file by making it more local to the place which requires it. Altough the term replica management may be pertinent to several areas distributed computing research, we deal mainly with the Grid and Grid computing paradigm. For the Grid, a replica management suite has been developed in the European DataGrid Project[13]. The software developed in work package 2 of the DataGrid[14] covers the registration, lookup, transfer and replication tasks of a mature replica management suite, with sufficiently distributed control. Its last implementation is based on the modern paradigm of web services and OGSA[11] architecture. Anyway, it is lacking a modern and scalable metadata repository.

3.6 Metadata Production and Storage

The metadata production and storage tasks are handled by three different components of the FloodGrid data management system. Metadata is produced by a set of specialised programs and scripts, which extract important values from datasets (both transferred from outside of FloodGrid and computed). The metadata is then stored in the metadata database - a RDBMS, interfaced with the rest of the FloodGrid via the metadata service. While we find it useless to describe the metadata extraction methods, which vary from one type of dataset to other and are not very complicated in general, we focus our view on the two remaining components of FloodGrid metadata system - the service and database.

3.7 Metadata Service

The metadata service (named org.crossgrid.wp12.metadata.service) is an OGSI-compliant[12] web service. It enables its user to add, remove, edit metadata descriptions of files (identified by a GUID) as well as to find registered files with certain properties. The service interface exports these methods:

- AddObject
- RemoveObject
- ModifyObject
- FindObject
- ShowStructure

The **Object* methods allow the user to work with metadata description of a file - he/she may add a description, remove or change it or find a file (a set of files) by its (partial) description. The *ShowStructure* method is not connected to a file, but rather to the whole database. Because the system is modular and the metadata service can be used to access any database conforming to some rules (described below), this method is necessary for the service user to properly display stored data. This method shows all available metadata items, their types and allowed or available values (in case of enumerations).

The service is accessible either by a client library,available in the Java 2 programming language, or by a visual interface implemented in the FloodGrid portal.

3.8 Metadata Database

The metadata database is a structure, which supports:

- Metadata items of types *String*, *Integer*, *float*, *date* (datetime) and *geometry* - geometrical shape (point, line, rectangle, polygon).
- closed enumerated sets
- open (modifiable) enumerated sets.

The structure of the database is not hardwired into the metadata system; it is defined in a table and can be modified anytime. Simple restart of the metadata service is then needed to access the new structure. Enumerated sets are implemented via indirection and are very useful for example for string values, where frequent use of identical values is expected (like names of users, for example). Instead of storing multiple copies of the same string, only a reference to another table - holding all the defined strings - is stored. In addition, such a set of values can be locked, so the user is forced to choose only from predefined values.

3.9 Typical Data Management Usage Scenario

To better illustrate the use of the metadata suite, described in this chapter, we wil present a usage scenario, which shows the coordination of single modules of the suite.

Let us imagine a case, where the user wants to simulate a weather prediction for certain time and area. He/she logs into the FloodGrid portal and starts the task by locating the input data for his/her simulation. He/she accesses the metadata lookup portlet, enters the file description (in this case type of file - ALADIN or MM5 input data, date and geographical location of the data). The metadata lookup gives him one or more files. He accesses the files' descriptions and chooses the right one. He clicks on its GUID and a set of physical location URLs is displayed. He chooses one of these URLs for the job definition document. Second option (currently under development) would be to just enter the GUID in the job definition document. The job could then access the most convenient replica, depending on where it would be started.

After providing the job definition document via the workflow portlet (described elsewhere), the job can be submitted. Once the job is executed on a computational node, all input data is downloaded (see Fig. 5) and computation can begin. Produced output data is registered and uploaded to Grid storage (CopyAndRegister function of the edg-rm replica management tool). Metadata is extracted (using the metadata extraction scripts) and sent back to the workflow service for registration into the metadata service. The upload of metadata to the workflow service is necessary because the computational nodes are not equipped with the software necessary for metadata service access. The cycle is now closed - new data is in the Grid, annotated and ready to be found and used in another computational job.

4 Future Development of Data Management Software in FloodGrid

We mentioned several 'under development' pieces of software in previous chapter. For example one such future enhancement not only of the data management suite, but of the whole PSE will be better integration of replication software with simulation jobs. The user will not have to find and select the physical file URL, the software will do it automatically based on the chosen GUID.

Another modifications are expected in the metadata service. The current prototype is witout GSI security - this will change. Also a method of distribution of the (potentially widely used) metadata service is considered. More metadata services could cooperate in a transparent way - users would 'write' metadata to their local service, but lookups will be done in all available networked services. Support for more data types is also considered, altough to date such need has not arisen.

References

1. Foster, I. Kesselman, C., The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., 1999.
2. The Globus Alliance. Available on: http://www.globus.org/.
3. Foster, I., Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 11(2):115-128, 1997.
4. EU 5FP project CROSSGRID. Available on:http://www.eu-crossgrid.org.
5. Hluchy, L. Astalos, J. Habala, O. Tran, V. D. Simo, B., Concept of a Problem Solving Environment for Flood Forecasting. Recent Advances in Parallel Virtual Machine and Message Passing Interface. LNCS 2474, Springer Verlag, 2002.
6. The FloodGrid Portal. Available on:https://portal.ui.sav.sk:8443/flood/.
7. The International ALADIN Project. Available on: http://www.cnrm.meteo.fr/aladin/.
8. Grell, G., Dudhia, J., Stauffer, D., A Description of the Fifth-Generation Penn State/NCAR Mesoscale Model (MM5). NCAR/TN-398+STR. Available on: http://www.mmm.ucar.edu/mm5/documents/mm5-desc-doc.html.
9. Hydrological Simulation Program-Fortran. Available on: http://water.usgs.gov/software/hspf.html.
10. FESWMS - Finite Element Surface Water Modeling System. Available on: http://www.bossintl.com/html/feswms.html.
11. Foster, I. Kesselman, C. Nick, J. M. Tuecke, S., The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems Integration. Available on: http://www.globus.org/research/papers/ogsa.pdf.
12. Tuecke, S. Czajkowski, K. Foster, I., Open Grid Services Infrastructure 1.0. Available on: http://www.ggf.org/ogsi-wg.
13. The DataGrid Project web site. Available on: http://www.eu-datagrid.org.
14. Kunszt, P. Laure, E. Stockinger, H. Stockinger, K., Advanced Replica Management with Reptor. In 5th International Conference on Parallel Processing and Applied Mathematics, Czestochowa, Poland, September 7-10. Springer Verlag, 2003.

Adaptive Task Scheduling in Computational GRID Environments

Manuel Hidalgo-Conde, Andrés Rodríguez, Sergio Ramírez, and Oswaldo Trelles[1]

Computer Architecture Department, University of Malaga,
Campus de Teatinos, E-29071 Malaga, Spain
{mhc, andresr, serr, ots}@ac.uma.es

Abstract. In this work we present the design and development of an adaptive task scheduling model which enables the definition and exploitation of a framework especially suitable for managing environments of intensive computing load. The framework supplies queuing mechanisms, priority-based scheduling and resources allocation strategies, load monitoring, and implements fault tolerance procedures. Buffering strategies have been used to reduce idle time for load reposition and to take advantage of I/O overlapping, increasing efficiency in the use of resources. Several tests have been performed using applications from the bioinformatics domain for which adaptive strategies have shown their ability to produce a noticeable reduction on execution time.

1 Introduction

The spread of GRID technology has promoted the development of distributed frameworks for high performance computing, by means of the intelligent integration of disperse and heterogeneous computational resources through high speed networks. Nowadays, GRID computing [1] most probably represents the new generation of web-based technology.

Grid technology began to be used in the middle 90's to reference a distributed computing environment oriented to solve heavy CPU tasks in science and engineering [2]. The Grid provides consistent and low-expensive access to installed computers with dramatic impact on CPU-power capabilities. Access to advanced computational capabilities allows broad classes of new applications to emerge, and to carry out significantly large tasks opening up new capabilities for knowledge generation.

This new computing capability becomes of special interest in the bioinformatics application domain. Current bioinformatics is mainly characterized by: (a) services and data are frequently replicated in several servers; (b) data collections are impressive and grow at exponential rates nurtured by technological breakdowns such as sequencing and gene expression monitoring technologies; (c) the park of installed hardware in traditional wet labs is traditionally conformed by collections of PCs. Thus Grid technology appears as the natural alternative to provide computational power in this scientific community.

[1] To whom correspondence should be addressed.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 880 – 890, 2005.

Nowadays, Globus [3] is the most used middleware for computational Grids implementation (*i.e.* NSF´s Teragrid [4], NASA´s Information Power Grid [5], etc.). Condor [6, 7] is another good alternative mainly focused in distributed owned computing resources. In the bioinformatics arena IRISgrid [8] propose a national bioinformatics Grid in Spain, and myGrid is the equivalent European proposal which includes a nice prototype —Taverna [9] — to facilitate the description of workflows.

Several proposals have addressed the task scheduling problem in Grid environments. Nimrod-G [10] issues the distribution of multiple runs of the same process with different parameters, under static Grid configuration, not being able to incorporate new resources when they become available. This problem is solved by GRaDS scheduler [11] by adapting dynamically the strategy when new resources are available in the Grid, using information continuously received from the grid nodes. Finally, Condor-G modifies the standard Condor scheduler to incorporate new machines —external to the Grid— connected by Globus. None of these schedulers incorporates fault tolerance concerns, neither dynamic load balancing as a function of the real capacity of each component of the Grid, issues, both, that are the central aspects of this work.

Several exercises have been implemented as initial set of experiments to demonstrate the ability of the proposal to adapt load distribution and support fault tolerance, and to evaluate the effectiveness of the scheduling strategy.

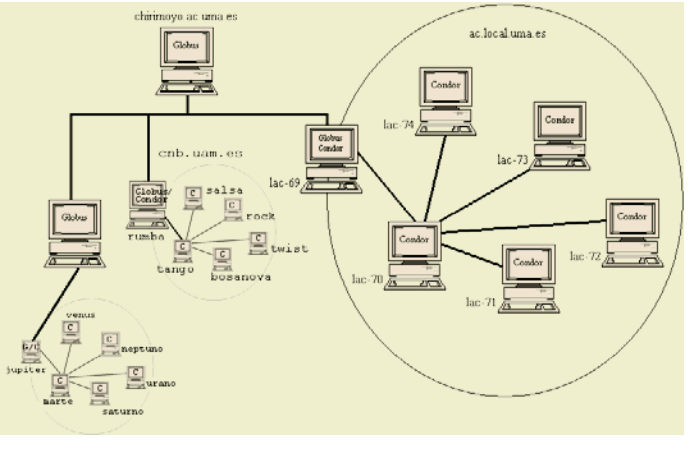


Fig. 1. General architecture of the Grid, formed by several nodes and a master service. The internal management of each cluster is performed with Condor, and a Globus/Condor interface allows the control of clusters

2 System and Methods

The Grid model we have devised has the following characteristics:

1. The general Grid architecture is composed of several nodes, which at the same time can form an internal sub-Grid (see Figure 1).

2. Tasks to be solved are queued in a pool from which the scheduler has access to them. Tasks arrive to the queue from the upper module which is able to solve data dependencies (identified by the use of the same file names) between tasks.
3. Our model is able to distribute a given task between different nodes available in the Grid. To perform this task the model use a specific splitting-module associated to the task. Un-divisible tasks or tasks that do not have the associated splitting-module are considered as unitary tasks.
4. The scheduler maintains a dynamic control of the Grid configuration, being able to incorporate new resources at any moment.
5. Fault tolerance mechanisms allow the detection of machines which do not respond to the assigned task. Under this scenario the task is re-scheduled.
6. Dynamic task distribution is governed in a by-demand fashion. The system maintains a record of the quality level of the service (response time) used to fine-tuning the distribution parameters.
7. Idle time —for load reposition— is avoided by sending new tasks in advance (buffering strategy)

The model has been implemented as follows (see Figure 2):

1. Globus and Condor have been used for inter- and intra-nodes connection respectively.
2. Workflows pre-processing module.

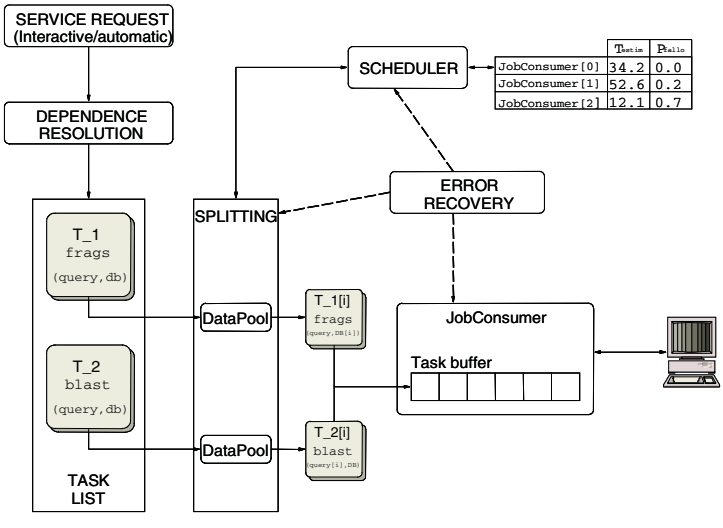


Fig. 2. The System Architecture is shown. Work requests are analyzed to solve data dependencies and divided in individual requests. The scheduler works over this pool of tasks and uses a map of services. When available, a "splitting" process can be applied over large tasks. The configuration-module dynamically traces resources. It works in close collaboration with the fault tolerance module who is in charge of the pending tasks, and it is able to re-insert tasks to be re-scheduled

3. Splitting tasks module.
4. Configuration module.
5. Fault tolerance module.
6. Scheduling module (dynamic on demand).
7. Buffering module for in-advance tasks.

The proposed Grid scheduling architecture is shown in Figure 3. As can be observed, the scheduler distributes tasks (Jobs) to the different JobConsumers (JC), each of one representing an available machine. This machine can be a node, with Globus external interface for a Condor cluster. In this case the interface transfers the job to the internal node. The JC maintains the record of the pending jobs and the machines on charge of it. Since a JC can represent a whole node, a buffering technique has been implemented, to send work in advance. Each JC uses a buffer as large as the number of jobs that can be active in a given instant for that JC. When a given job ends its position in the buffer becomes free and a new job can be launched without delay for job reposition from the scheduler.

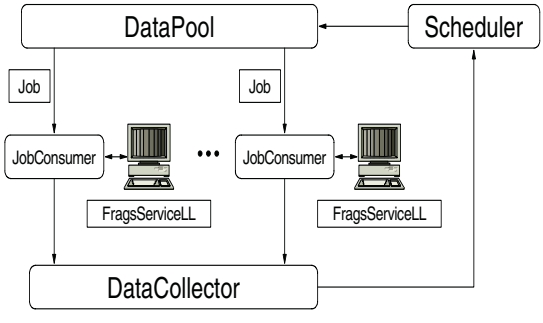


Fig. 3. The scheduling module architecture. The DataCollector module, in charge of recovering the task-output, informs the scheduler with the statistics of the task. The scheduler drives the DataPool module to distribute the load (biological sequences in this case) to the JobConsumers

Although we have been using the buffering technique since our first works in multiprocessors [12, 13], in this case we have additionally observed the employment of I/O overlapping in the same machine, which is in fact important because bioinformatics applications are mostly I/O bounded.

The scheduler can work in two different ways: (a) for un-divisible or atomic tasks (tasks of reduced CPU cost or tasks only available in this model) or (b) it can be assisted by a process in charge of dividing tasks (*splitter*). This case is becoming frequent for very long tasks in bioinformatics (i.e. genomes comparison that can be divided at gene level comparison). In this case, a *splitter* process could divide the data base (genome) in different sectors to be distributed to the JC.

Whenever possible, sending large datasets through the network has been avoided due to the high impact in efficiency. Since datasets in bioinformatics are frequently replicated in the servers, the scheduler does not send the sequences but an identifier

(in general, the database name, and pointers to define the partial set of data to be processed; i.e. "process *n* sequences from the database *x* starting at point *i*").

The scheduler module also maintains the state of the configuration. In particular, it is important to detect inactivity in the service with pending tasks. In this case, their tasks are re-inserted in the task-pool (at the beginning) as a whole (unitary) tasks. New machines are simply inserted in the node-pool and immediately become available to receive jobs.

Load distribution is performed in a dynamic and adaptive fashion as fast as new tasks are demanded by nodes. The current configuration is used to know the computational power available at that time. The system evaluates the CPU cost of each task and adjusts predictions when the tasks are reported from the services. Load size is computed as a function of the tasks CPU cost and the quality of the service, estimated as the historic response time (a configurable parameter).

With some more detail the load distribution is estimated as follows: Using the splitting strategy, a heavy task (such as the querying by content of a DNA database) is divided in a number of sub-tasks. The length of the task is determined as a number of sequences involved in the task (atomic units) and is computed dynamically as a function of the available resources in each JC.

To determine the load allocation for a given node the JC records information about the number of atomic tasks sent to the nodes. When a process ends and sends the results back to the corresponding JC, it is able to detect the response time. The JC send this information to the DataPool and to the Scheduler, who maintain the updated record of the average speed observed in each service (in units/sec) corresponding to each JC (v_i), and the global average speed in the system (v_{global}).

When a given JC requests more data to the DataPool, the DataPool asks to the scheduler the optimal number of atomic units to be sent to the JC. The scheduler computes this value (Q) based on the initial task size sent to the JC (Q_{inic}), the average speed of the JC and the global average speed of the system:

$$Q = \frac{v_i}{v_{global}} Q_{inic} \tag{1}$$

This distribution scheme benefits those JobConsumers which response is faster than the global average velocity ($v_i > v_{global}$), increasing the number of units to be distributed to them. On the other hand, for JobConsumers under the average velocity ($v_i < v_{global}$), the number of tasks to be processed goes down.

To compute the average speed of execution (in units/sec) the recent history is given more weight than the old one. This average speed is computed in the JobConsumer (at the end of the completion of every task), as:

$$v_i = v'_i \cdot d + v_{exec} \cdot (1 - d) \tag{2}$$

where v_i correspond to the new average speed, v'_i is the previous value and d (decay) is a configurable value (by default is 0.8) related to how fast the older values are forgotten by the system, and thus it controls the velocity of adaptation of the scheduler to the changes in the quality of the service (measured as response time).

3 Results

In this section, we present the initial results of the proposed implementation.

3.1 The Framework

The general architecture used on the tests is composed of the following nodes:

lac-69.ac.local.uma.es A cluster of five Pentium IV 1.7 GHz machines with 128 Mb of RAM memory, belonging to the educational laboratories of the Computer Architecture Department at The University of Malaga. These laboratories are in a domain that is different from the other machines (`ac.local.uma.es`). A Globus interface will be installed in one of the machines (`lac-69`), which will distribute the jobs to the other machines through Condor.

chirimoyo.ac.uma.es A server with two Intel Xeon processors, Hyperthreading technology, 2.4 GHz and 1 Gbyte of RAM memory.

cierzo.ac.uma.es A Pentium III workstation, 1 GHz and 256 Mb of RAM memory.

tierra.es A Pentium II 233 MHz workstation with 186 Mb of RAM memory, placed outside Malaga University's domain and at about thirty kilometres from it.

On these computers, Globus Toolkit version 2.4 (http://www.globus.org) has been installed. Globus together with the Java CoG kit (http://www-unix.globus.org/cog) enables us to make the most of the grid possibilities for the submission of jobs and the secured and authenticated data transfer between nodes.

In the educational laboratory, Condor toolkit (www.cs.wisc.edu/condor) has been installed. The node lac-69 will have the Globus Toolkit installed together with the jobmanager-condor. Thus, jobs arriving through Globus will be sent to the Condor scheduler. The configuration used in the tests is described in Figure 1 and is equivalent to a heterogeneous network with clusters scheduled by Condor and central control under Globus.

3.2 The Application

Bioinformatics deals with applying information technology to the sciences of life. On these, high performance technologies development has produced the appearance of data at a growth rate that was not even imaginable before. This has opened a new behavioural paradigm on this area that tries to approach the analysis of whole organisms (genomes) instead of genes or isolated products. This fact takes to a disturbed growth of computing necessities.

On the other hand, this kind of massive analysis needs the combination of multiple tools in a workflow. These require of diverse data collections available in specific places. Thankfully, this model of workload is perfectly projected on a distributed computing structure, as the majority of the problems are divisible in big high-computational-demand process and independently executable (very gross grain).

Among the implemented applications, we will describe the searching of similar fragments between sets of biological sequences. This is an application that has been

developed in our installations [14] and is suitable for the use of a splitter that divides the global task in several partial tasks and then must integrate the results. This application will include the particular cases of indivisible applications like proprietary code —BLAST [15]— that would not be subject to modification.

The Frags procedure implements a mechanism of data mining in biological sequences and is oriented to the searching of low-level signals (short sequence fragments) that are present in several sequences. This accumulation is what strengthens the signal and gives it a biological sense.

The application compares one query sequence against all the sequences present in a database via an exhaustive signal searching (see Figure 4). A signal is defined as a sequence fragment that is present in the query sequence and any sequence of the database and reaches a maximum score (based on a residue-similarity table).

Our implementation receives one file of query sequences that must be compared against the whole database (see Figure 5). The results of this signal searching are used in the elaboration of association rules which main utility is the prediction of protein functions [16].

In the infrastructure offered by the Grid, the Frags service is presented as a service accessible through a call to an executable program (fragsxml.exe) that receives an XML document in the standard input with all the necessary parameters for Frags execution (including beginning and count in database, IniDB and FinDB), as well as the query sequences to what the service is required to be applied. In case the server does not have a local copy available, the sequences of the data base can be obtained through a new service call (getSequences) to another node of the grid (see Figure 5). In the current configuration, only the computers in the cluster with Condor have the database locally installed. The rest of the machines will have to access to a new service to obtain the sequences to process with the corresponding associated delay.

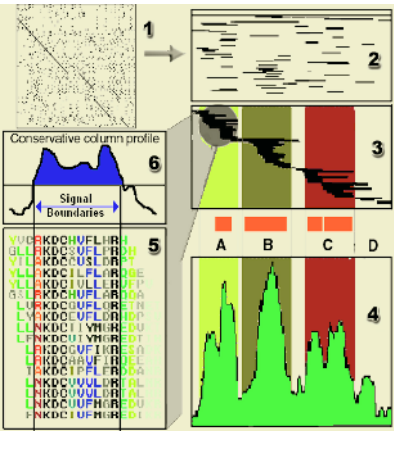


Fig. 4. Signal searching strategy used by Frags: (1) when comparing two sequences local fragments show the similarity between the sequences; (2) these small fragments are organized by their position in the query sequence and (3) are ordered. A segmentation process allows detect the accumulations (4) and the border refinement is applied in (5) and (6) to obtain the signal

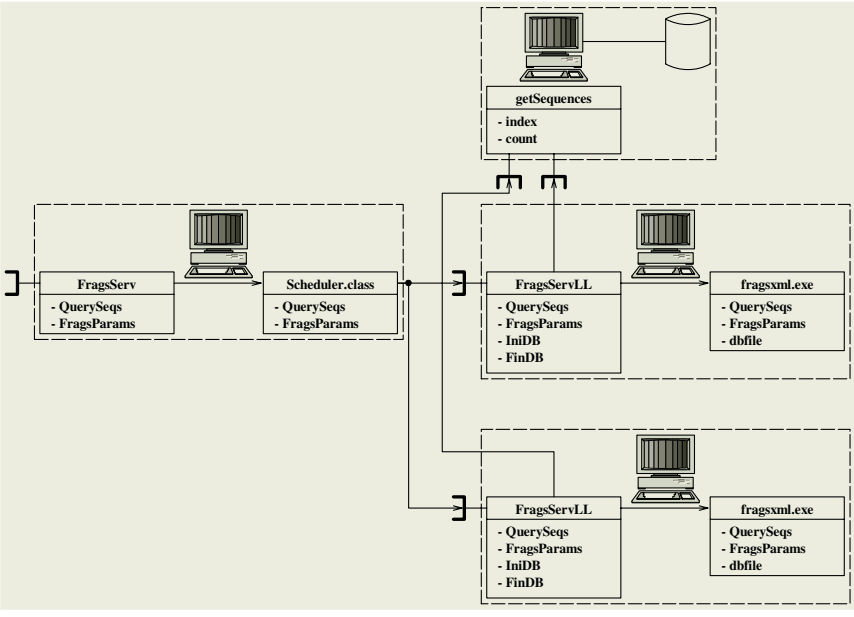


Fig. 5. Sketch of Frags service in the grid. The servers implementing the Frags service can access the other machines in the Grid to obtain the sequences to process from the database without having to neither transfer nor store the whole database

Next, the results obtained in several tests based on the execution of the Frags service will be presented. First, we studied the effect of the in-advance delivery of jobs (buffering). We employed a database with 8000 sequences (one tenth of the real case) and divided it in equally sized blocks of 125, 200… sequences. Next, the number of jobs sent in advance has been taken as a parameter and response times have been measured. To be able to detect the results of this experiment in a better way, only one of the computers has been used (the server chirimoyo).

In Figure 6 –in the left- the time spent to process the database varying the buffer size is represented. As it can be noticed, the execution time is significantly reduced when we increase the buffer size as the effect of at least three causes: (a) the removal of the delay for task waiting; (b) the advantage of concurrent task execution with a strong component of input-output and (c) a lesser cost associated to the number of tasks to distribute (controlled by the block size). The decrease has a limit in which the increasing of the buffer size does not mean any improvement due to the overload that this causes to the operating system.

Next, we studied the effect of the initial load size, that has influence on the service set-up time and on the time the scheduler needs to adapt to the components behaviour to reach the optimum load distribution. In Figure 6 –right hand side- this influence over the total execution time is showed. In this case, the parameter is the number of sequences sent to each node on the first executions.

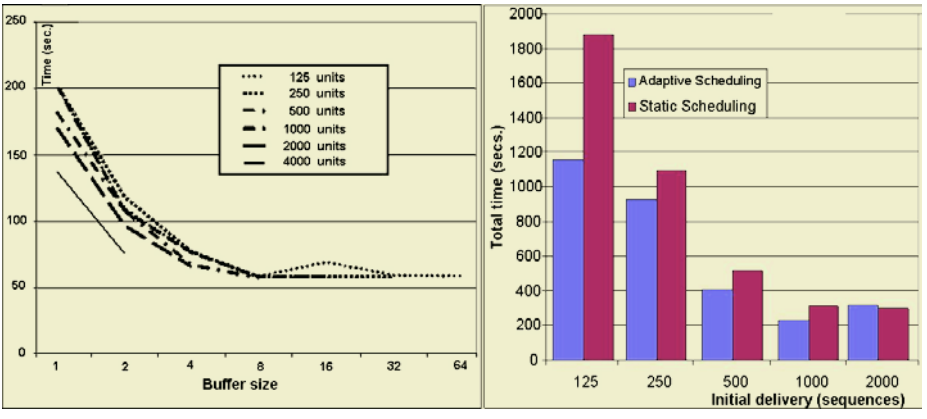


Fig. 6. In the left, the execution time for the Frags application over a 8000 sequences database against the buffer size used and using blocks of different size. Shorter lines are due to a lesser number of available tasks as we vary the block size. The computer used (chirimoyo) is a two-processors computer, for this reason the first reduction is due to the use of the second CPU, while the reductions observed for 2 and 4 tasks can be associated to the overlapping of the I/O. The reductions —lighter— when reaching 8 tasks can be associated to the removal of the waiting time for load stabilization. In the right hand side, the Influence on execution time of initial task size. First, a better response time of the proposed scheduler as the load is nearer to the optimal load can be observed. Second, adaptive scheduling always has a better performance than fixed scheduling. For very little initial loads the execution time grows due to the delay introduced by communications

Accordingly, wheater the initial load is too small, the scheduler looses much time in communication with nodes. For very big initial load, the scheduler does not have time enough to adapt to the configuration and the slower nodes worsen the system performance. Comparing these results to the results obtained using a non-adaptive scheduler, the smaller the initial load, the bigger the improvement obtained with the adaptive scheduler. In the optimal case of a 1000 sequences initial load, the adaptive scheduler spent 230 seconds to process the whole database while the faster computer in the grid spent 1086 seconds.

The next step is checking the behaviour of the scheduler in the processing of a database of a larger size (8000 sequences). The time spent by the faster computer in the grid was 1086 seconds. We will see how this time can be drastically reduced by the adaptive management of the scheduler.

The scheduler will adapt the load distribution according to the response times of each node in the grid. As it can be appreciated in Figure 7, the response of the cluster lac-69 (composed of five workstations with a local copy of the database) receives more load due to the fact that its response is faster that the other nodes. On the contrary, the node ‘tierra’ receives little load as it presents a much lesser yield than the rest of the nodes and could worsen the global system’s performance.

To test the fault-tolerance module, network breakdowns have been simulated by the use of *timeouts* —if a task takes too much time, the task is aborted—. This causes

the effect that can be appreciated in Figure 6, where the processing of the fragments of the database that could not be processed in other nodes produce the falls that can be appreciated in lac-69's graph.

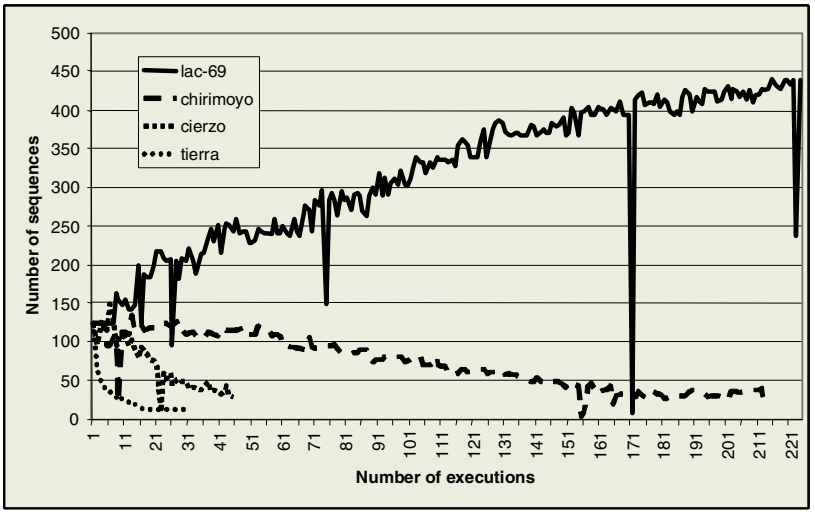


Fig. 7. Number of executions and size (number of sequences) of the task sent by the scheduler for the nodes lac-69, chirimoyo, cierzo and tierra. The strong falls that can be observed in lac-69's graph are due to the fact that the error-recovery module assigns the sequences that could not be processed in other node to the first node that becomes available

4 Discussion

We have presented an adaptive scheduling model for the grid including task splitting and being able to adapt to the resource availability, with the capability of detection of new nodes as well as the erroneous operation of the present nodes. This scheduling policy has been applied to the resolution of a biologic problem on the bioinformatics area as it is the searching of common fragments between a set of query sequences and a data base.

Several experiments have been carried out in order to analyze each strategy component individually and to evaluate their influence on the efficiency improvement. The results obtained when adjusting parameters such as task buffer size, task size and initial number of tasks to distribute have been presented. These results have been compared to the results obtained using a static scheduling scheme with very encouraging results.

Acknowledgements

This work has been partially financed by the Genoma-España Foundation in the project GV5-Bioinformática Integrada of the Instituto Nacional de Bioinformática.

References

1. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. Supercomputer Applications, 2001. http://www.globus.org/research/papers/anatomy.pdf.
2. Foster, I. and Kesselman, C. (1998). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1 edition.
3. Foster, I. et al. The Globus Project: A Status Report. In Proceedings of the Seventh Heterogeneous Computing Workshop (1998) (www.globus.org)
4. TeraGrid: http://www.teragrid.org/userinfo/docs/TeraGrid-Data-Primer.pdf
5. NASA information power Grid Project: http://www.globus.org/presentations/retreat98/power_grid/
6. Condor: High Throughput Computing (http://www.cs.wisc.edu/condor)
7. D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, Grid
8. Área temática de bioinformática, 2003. Available at http://irisgrid.rediris.es/doc/biogrid.pdf.
9. R. Stevens, A. Robinson y C. A. Goble. myGrid: Personalised Bioinformatics on the Information Grid. In proceedings of 11th International Conference on Intelligent Systems in Molecular Biology, 29 June - 3 July 2003. Brisbane, Australia. Bioinformatics Vol. 19 Suppl. 1 2003. (http://www.ebi.ac.uk/mygrid/)
10. Abramson, D. et al. (2000) “High performance parametric modelling with Nimrod/G: Killer application for the global Grid?”. In proceedings of the Seventh International Symposium on High Performance Distributed Processing Symposium (IPDPS’00)
11. Berman, F.; et al. (2001). The GrADS Project: Software support for high-level Grid application development. International Journal of Supercomputer Applications 15, 4 (2001),327-344.
12. Trelles-Salazar, O.; Zapata. E. and Carazo, J.M., (1994) “Mapping Strategies for Sequential Sequence Comparison Algorithms on LAN-based Message Passing Architectures”, High Performace Computing and Networking (HPCN-Europe'94), Munich-Deutschland
13. Trelles-Salazar, O. Zapata, E.L. and Carazo J.M. (1994), “On an efficient parallelization of exhaustive sequence comparison algorithms”; Computer Applications in BioSciences 10(5):509-511
14. Rodriguez, A. Pérez-Pulido, A:, Thode,G.; Carazo,JM. y Trelles,O. (2000); “Mining Low-level similarity signals from sequence databases”, High Performance Computers Applied to BioInformatics and Computational Biology. 4th Conference on Systematics, cybernetics and Informatics, SCI'2000; Orlando, Florida, USA.
15. Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ.; (1997), “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs”; Nucleic Acids Res. 1997 Sep 1;25(17):3389-402
16. Pérez, A.J.; Thode, G. and Trelles, O. (2004); “AnaGram: protein function assignment”; Bioinformatics Vol. 20 no. 2 2004, pages 291-292

Large-Scale Computational Finance Applications on the Open Grid Service Environment

Ronald Hochreiter, Clemens Wiesinger, and David Wozabal

Department of Statistics and Decision Support Systems, University of Vienna

Abstract. In this paper we review the central concepts of the Open Grid Service Environment, which represents an abstract service stack and was introduced to design workflow-based problem solving environments. So far, it has been used to model large-scale computational finance problems as abstract workflows with meta-components and instantiate such workflows with different components based on semantic matching. In this paper we continue by presenting two further examples from the field of computational finance, which substantiate the need for Grid technologies and take a closer look at the implementational issues and apply the concepts of Enterprise Service Busses for parallel process orchestration. The CCA (Common Component Architecture) and its convergence to Web service specifications is the key technology for integrating heterogeneous components, with which one has to deal with in the financial sector.

1 Introduction

In a recent paper [1], we provided an overview of the next generation technology for the adaptation of the AURORA Financial Management System [2], which is a component-based computational financial management system allowing for solving many real-world - hence large-scale - investment problems. While in earlier papers, we focused on the modeling aspect of financial decision problems, we are now presenting solutions for the implementational issues of such large-scale financial workflows. The AURORA sub-project *High Performance Computing in Finance* is a part of the AURORA project (Advanced Models, Applications and Software Systems for High Performance Computing) funded by the Austrian Science Fund (FWF), whose research focus has been shifted to Grid computing during the last years. The core of the AURORA project is actively involved in various Grid projects and provides an ideal infrastructure for application groups to Grid-enable applications and thus being able to instantiate huge real-world models and solve problems of unprecedented size and complexity.

In a recent article in Grid Today [3], it is argued that the financial services sector will be a driving force in the application of Grid technologies. Especially the recent additional set of regulatory constraints for the financial industry, like Basle II, forces large financial companies, like BNP Paribas, to create their own Grid applications. In this context, the AURORA Financial Management System has also been adopted to make use of the available Web- and Grid technologies.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 891–899, 2005.

This paper is organized as follows. In section 2 we review the most important facts of the Open Grid Service Environment presented in [1], which represents the basic concept, on which the implementational issues described in this paper are based upon. Section 3 provides two examples from the field of operations research and computational finance, namely large-scale portfolio selection and asset liability management. Both problems are generally solvable on clusters, however, we present extensions, which necessitated recently, where a Grid solution is inevitable to calculate numerical results computationally in reasonable time and quality. Section 4 describes the enabling technologies, namely the Common Component Architecture (CCA) and the Vienna Grid Environment (VGE). Section 5 concludes the paper with a summary and an outlook of future developments.

2 Open Grid Service Environment

In [1] an abstract framework of a distributed system for use in computational finance was proposed. The core of the considerations given in that paper is an integrated service stack (the OGSE Service stack), which is an extension of the W3C Web Service Stack (see figure 1). The view on the service stack is application-centric from top down and Grid-centric from bottom up. Application-centric services support wrapping existing code into components and deal with problem specific data. Grid-centric services ensure the compatibility to underlying low level network facilities and handle workflow enactment on specific computational resources as well as discovery and allocation of these resources.

Though the system is designed to be used as an financial management system it can act as an unified framework for a broad range of application classes, such as decision support systems for e.g. Power and Energy Systems and Supply Chain Management.

The system is designed to integrate and orchestrate software of different origins over a network/Grid. The atomic unit of the system is a component. Components exist as

- abstract component descriptions (called meta-components), which clearly define input and output parameters,
- and concrete (interchangeable) implementations, which follow these (meta) specifications.

To run a task, the user (e.g. a financial engineer, more generally a decision taker) orchestrates different components to a workflow. Workflows and components exist in abstract and concrete form. Meta-workflows consist of general meta-components and act as abstract problem descriptions. Concrete instantiations of meta-workflows are workflows of components matching the meta components specified in the abstract description of the workflow. By this way it is possible to formulate different concrete workflows that match one specified meta-workflow, if various concrete components exist.

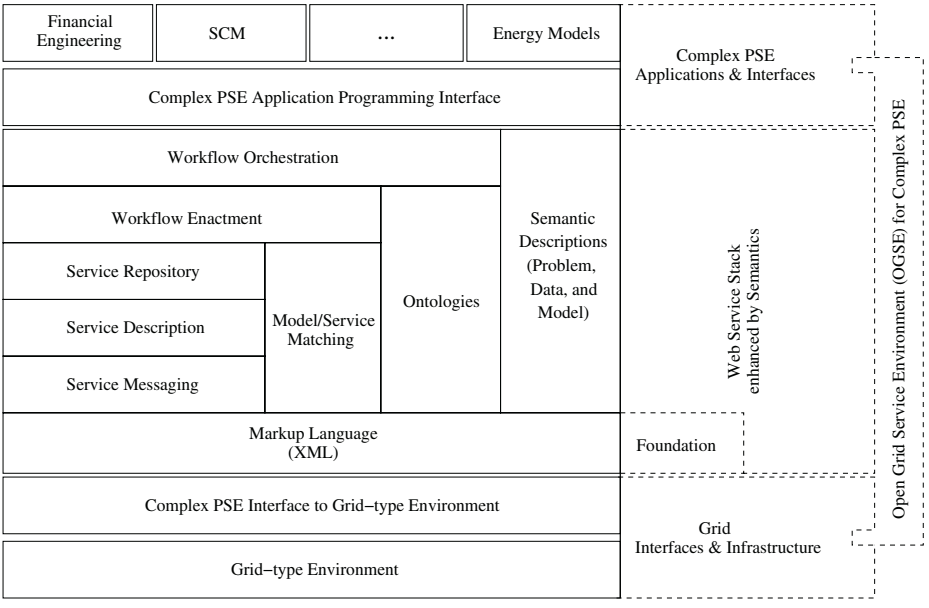


Fig. 1. OGSE Stack for complex problem solving applications

The flexibility of such a system originates from the interchangeability of implementations through the high level description of components and workflows in the meta-layer. Real-life examples substantiating that such a system supports the needs of financial engineers for accomplishing complex tasks in the field of financial modeling have been presented in [1]. The usage of this abstract and concrete information supports

- a semi-automatic component selection and orchestration,
- and a matchmaking service ((financial) model/service matching), which enables the user to specify a problem abstractly and receive a proposition of a possible solution to the respective problem in the form of a concrete workflow.

In this paper the focus is laid on describing the component and Grid architecture of the system and presenting examples that illustrate and motivate the Grid design decisions.

3 Large-Scale Computational Finance

As already indicated in the introduction, the area of Financial Engineering is one of the driving forces of Grid technologies. We consider the sub-area of investment management and take a closer look at two rather different instances of this

sub-area. In both cases the financial modeler aims at representing the future uncertainty of the real world as close as possible in her financial model. The more realistic the model becomes, the more computing power is necessary to obtain valuable decisions in reasonable time. Although even huge singular problems can be solved on rather small cluster computing infrastructures, inventions of new complex iterative techniques as well as regulatory constraints imposed on financial companies motivate and necessitate the use of Grid technologies.

Another issue is that the set of (financial) data used for calculations of optimal decisions is getting larger. While at the beginning of computational portfolio management - during the 1960s and 1970s - monthly historical data was considered, it is common to use even minute-based data e.g. for high-frequency intra-day trading.

3.1 Single-Stage Auto-Selective Portfolio Management

In the first case we consider the well-known single-stage portfolio investment management problem, which was introduced in the early 1950s by Nobel-laureate H. M. Markowitz [4]. The basic task is to solve the bi-criteria optimization problem

$$\begin{array}{ll} \text{maximize in } x: \mathbb{F}(x,\xi) = \mathbb{E}(x,\xi) - \lambda\rho(x,\xi) \\ \text{subject to} \quad \mathbb{E}(x,\xi) \geq \mu \\ \qquad\qquad\quad x \in \mathcal{X} \end{array} \tag{1}$$

i.e. finding an optimal portfolio x by maximizing a functional $\mathbb{F}(x,\xi)$ of the expected wealth $\mathbb{E}(x,\xi)$ and the inherent risk $\rho(x,\xi)$ of the portfolio. $\mathcal{X}$ represents a set of organizational and regulatory constraints, while μ is the minimal expected return requested by the decision maker. Due to the fact that the decision depends on the (uncertain) developments of the specific assets represented by a random variable ξ, such problems can become computationally intractable when realistically many possible realizations of (uncertain) data is considered. The constraint set $\mathcal{X}$ as well as specific risk measures $\rho(x,\xi)$ may additionally induce non-linearities and/or even non-convexities into the optimization problem, which require computationally intensive heuristic solution techniques to finally solve the respective problems.

Even when extending these problems to the two-stage case [5] or if a huge set of data is considered, such problems are solvable on a reasonable sized cluster. In many cases there exists no real decision basis for selecting the underlying risk measure $\rho(x,\xi)$. For that reason, adaptive methods have been proposed, which select the currently best risk measure from a set of risk measures $\mathcal{R} = \{\rho_1(x,\xi), \rho_2(x,\xi), \ldots, \rho_n(x,\xi)\}$ automatically. The choice is basically based on the analysis of the short-term history of underlying uncertain factors (asset prices, interest rates, ...). However, even for a reasonable small set of different risk measures, a complete backtesting procedure requires an enormous amount of different optimization runs, most likely with different optimization techniques and a strict control of the workflow, i.e. the optimization results have to be compared and a decision has to be incorporated into subsequent selections.

The application of such a methodology clearly raises the need for a Grid solution, because the workflow for solving such a complex iterative process involves the execution of many heterogeneous algorithms, which can be spread over clusters or other similar (network) architectures.

In Figure 2 the Parallel Process Workflow Variation pattern is applied to this single-stage auto-selective portfolio management problem schematically. In chapter 4 the implementational structure, to which such components can be added, will be described.

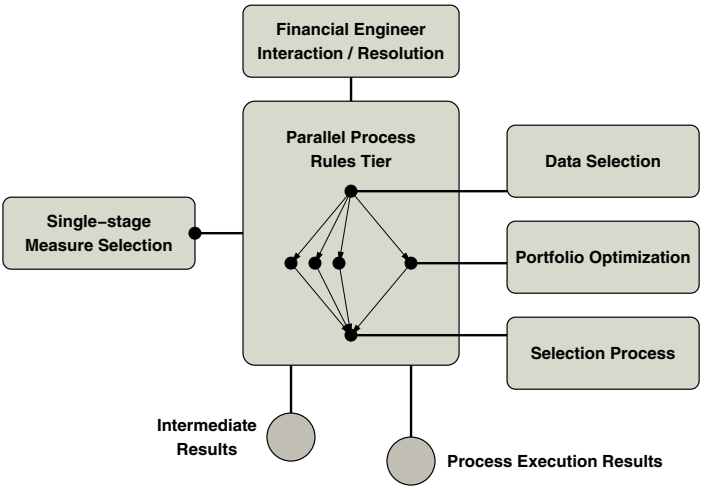


Fig. 2. Parallel Process Workflow Variation Pattern: Automatic Risk Measure Selection

3.2 Multi-stage Stochastic Asset Liability Management

In the second case, we focus on multi-stage stochastic asset and liability management problems. A theoretical treatment of the underlying class of optimization problems can be found in [6]. Such problems have successfully been solved over the Grid in [7]. However, recently such models have been applied to value single pension fund contracts for insurance companies. If the company is aiming at solving and optimizing an enterprise-wide investment strategy, every contract has to be calculated. For a large insurance company, tens of thousands of contracts have to be optimized, while communication is still necessary in order to summarize the results for generating the overall investment strategy. See Figure 3 for an outline of the workflow from data to decisions. There is also a tight interplay between financial modelers and the decision taker, which is important from a decision support system perspective.

Still, these calculations can be conducted in parallel to a large extent. Although the underlying model - and the problem in general - is different compared

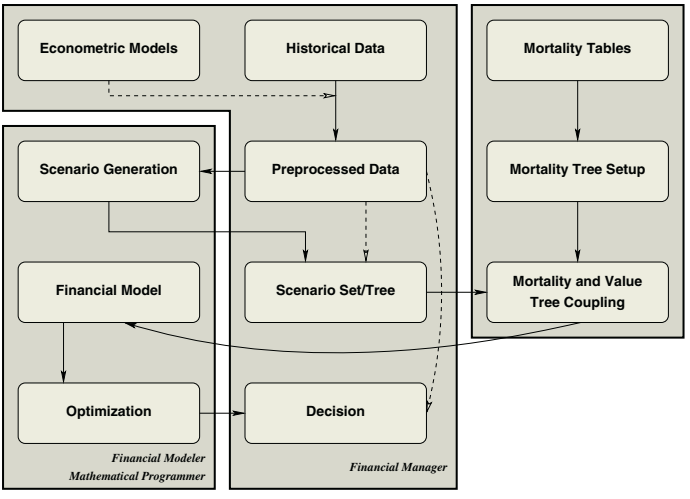


Fig. 3. Pension-fund optimization workflow - From data to decision

to the auto-selective portfolio management from a financial engineering point of view, the implementation applying the Parallel Process Workflow Variation pattern like in chapter 3.1 looks quite similar from a computer science point of view.

4 Implementational Issues and Technology Convergence

A problem solving environment (PSE), such as the AURORA Financial Management, often consists of component technology, due to the fact that such a system allows users to bring together a variety of computational tools to solve costly computational tasks in a domain-specific application. In general [8], a component is an independent unit of code that follows behavior rules and implements interfaces (ports). Additionally, components require an integration framework which is an implementation of a plug-in like environment for component interaction including creation, connection, manipulation, and destruction. The component architecture itself describes a common specification of a set of interfaces and rules for a component-to-component and a component-to-framework interaction. The Common Component Architecture (CCA) Forum is currently specifying and developing such a standard component architecture for high-performance computing, which matches our requirements for a component-based high-performance application in finance in many ways. Basic features that drove the decision to adopt the CCA are:

- the minimalistic and extensible view on CCA components.
- SIDL [9] as a programming-language-independent interface definition language used to describe component interfaces. Component descriptions using

SIDL can be used by repositories and by a proxy generator to provide the component stubs element of communication ports.
- the language interoperability through the Babel project [10].
- the looser coupling approach between the caller (user) and callee (provider) component through ports which converges with the web service paradigm.

According to the CCA specification [11], the common component architecture consists of a slowly evolving core specification and a continuously expanding default ports section. Relevant core interfaces are an unique component identity (ComponentID), a "plain" Port (inherited by every default and developer defined port), a Component interface to set Services, and the Services construct to manage all the component's ports. Default ports include, amongst others, the GoPort to start the component-based workflow and a BuilderService to create, introspect, connect, and destroy components. Several implementations of the CCA specification exist, like Ccaffeine [12], XCAT3 [13] and SCIRun2 [14]. SCIRun2 supports distributed computing through distributed objects and suggests a meta-component model to integrate different component models. This concept is somewhat similar to the concept of the OGSE.

The granularity of components is dictated by the complexity of an application architecture and by performance considerations. As an example of complexity, frameworks may themselves appear as components in order to connect to components in other frameworks. As an example of performance considerations in financial workflow applications presented in section 3.1. Figure 2 describes the granularity of financial management components schematically, but it can be seen that such workflows ideally suit into such a structure.

The core of a component integration framework is the enterprise service bus (ESB) which supports/implements a service oriented architecture (SOA), see [15] and [16]. Such a solution should meet a minimum set of capabilities according to communication, integration, and service interaction found in [16].

For all underlying communication paradigms: message passing, RPC, notification, shared spaces, message queuing, and publish/subscribe and its differences in the three decoupling dimensions - time (no need for an active participation in the interaction at the same time), space (the interaction partners do not need to know each other), and synchronization (partners are not blocked while producing and consuming events and can perform concurrent activities) see [17].

Such a prototype solution of a service-oriented environment for software-components, that targets most of the Grid-infrastructural issues described above, is the Vienna Grid Environment (VGE) [18] developed by another sub-project of AURORA dealing with runtime systems for scientific computing. In short, the VGE is a service-oriented Grid infrastructure based on standard Web Services technologies that is developed in Java and requires Tomcat and Axis to host both infrastructure and financial services. Basically, the enabling of existing financial software on the VGE is pretty simple and requires an installation as described in the VGE user guide (see [19] and [20]). Thus, VGE services are used to encapsulate native applications.

A first prototype for the single-stage auto-selective portfolio management (as presented in Chapter 3.1) was implemented on the VGE. Due to the lack of a specific workflow engine, the two basic components (data selection and portfolio optimization with different risk measures $\rho(x, \xi)$ were tested without semi-automatic selection and iterative features. The first impression looks promising. Both components use heterogeneous technologies, i.e. data selection is based on Python scripts and e.g. $\rho(x, \xi) = \text{CVaR}_\alpha$ (Conditional Value at Risk [21]) optimization is modeled with MatLab/Octave scripts and solved with a dedicated Linear Programming Solver (SoPlex [22]), and it was possible to integrate them seamlessly into a common component system.

5 Conclusion

In this paper, we extended the scope of the Open Grid Service Environment from modeling concepts for large computational finance applications to concrete implementational issues. We took a look at two heterogeneous financial applications, whose implementation is homogeneous, due to the application of the mergence of the enterprise service bus and parallel process orchestration patterns. The enterprise service bus is an ideal framework for an implementation of service oriented architectures. Furthermore, the CCA Specification (Common Component Architecture), and its convergence to the Web service paradigm, as a standard component architecture for high-performance computing has proven to be successfully applicable for the implementation of large-scale computational finance applications on the Open Grid Service Environment. The complementary application of a service-oriented environment for software-components, like the VGE, enabled the development of a promising prototype for implementing the system. Our development of this prototype is intended to close the gap between the basic (single) usage of financial components and an user-controlled financial workflow as depicted in Figure 2. The completion of this task outlines the next step in the ongoing development of the AURORA Financial Management System.

References

1. Wiesinger, C., Giczi, D., Hochreiter, R.: An Open Grid Service Environment for large-scale computational finance modeling systems. In Bubak, M., Albada, G., Sloot, P., Dongarra, J., eds.: International Conference on Computational Science 2004. Krakow, Poland. Part I. Volume 3036 of Springer Lecture Notes in Computer Science., Springer (2004) 83–90
2. Pflug, G.C., Swietanowski, A., Dockner, E., Moritsch, H.: The AURORA financial management system: Model and parallel implementation design. Annals of Operations Research **99** (2000) 189–206
3. Harris, D.: Will financial services drive Grid adoption? Grid Today **3** (2004)
4. Markowitz, H.M.: Portfolio selection. Journal of Finance **7** (1952) 77–91
5. Mulvey, J.M., Vanderbei, R.J., Zenios, S.A.: Robust optimization of large-scale systems. Operations Research **43** (1995) 264–281

6. Ruszczynski, A., Shapiro, A., eds.: Stochastic Programming. Volume 10 of Handbooks in Operations Research and Management Science. Elsevier (2003)
7. Linderoth, J.T., Wright, S.J.: Decomposition algorithms for stochastic programming on a Computational Grid. Computational Optimization and Applications **24** (2003) 207–250
8. Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., Smolinski, B.: Toward a common component architecture for high-performance scientic computing. 8th IEEE International Symposium on High Performance Distributed Computing (1999)
9. Cleary, A., Kohn, S., Smith, S., Smolinski, B.: Language interoperability mechanisms for high-performance scientific applications. SIAM Workshop on Object-Oriented Methods for Inter-operable Scientific and Engineering Computing, Yorktown Heights, NY (1998)
10. Dahlgren, T., Epperly, T., Kumfert, G. Leek, J.: Babel Users' Guide. Center For Applied Scientic Computing Lawrence Livermore National Laboratory. Livermore, California, USA (2004)
11. Kumfert, G.: Understanding the CCA standard through Decaf (updated for CCA 0.6.1 and Babel 0.8.4). UCRL-MA-148390 (2003)
12. Allan, B., Armstrong, R., Wolfe, A., Ray, J., Bernholdt, D., Kohl, J.: The CCA core specification in a distributed memory SPMD framework. Concurrency and Computation: Practice and Experience **14** (2002) 323–345
13. Krishnan, S., Gannon, D.: XCAT3: a framework for CCA components as OGSA services. In: High-Level Parallel Programming Models and Supportive Environments. (2004) 90–97
14. Zhang, K., Damevski, K., Venkatachalapathy, V., Parker, S.: SCIRun2: a CCA framework for high performance computing. In: High-Level Parallel Programming Models and Supportive Environments. (2004) 72–79
15. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ (2004)
16. Keen, M., Acharya, A., Bishop, S., Hopkins, A., Milinski, S., Nott, C., Robinson, R., Adams, J., Verschueren, P.: Patterns: Implementing an SOA Using an Enterprise Service Bus. IBM Redbooks (2004)
17. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.: The many faces of publish/subscribe. ACM Computing Surveys **35** (2003) 114–131
18. Benkner, S., Brandic, I., Engelbrecht, G., Schmidt, R.: VGE - a service-oriented environment for on-demand supercomputing. In: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004). (2004) 11–18
19. Benkner, S., Brandic, I., Engelbrecht, G., Schmidt, R.: VGSE user manual. http://www.par.univie.ac.at/project/vge/ (2004)
20. Benkner, S., Brandic, I., Engelbrecht, G., Schmidt, R.: VGCE user manual. http://www.par.univie.ac.at/project/vge/ (2004)
21. Rockafellar, R., Uryasev, S.: Optimization of Conditional Value-at-Risk. The Journal of Risk **2** (2000) 21–41
22. Wunderling, R.: Paralleler und Objektorientierter Simplex-Algorithmus. PhD thesis, ZIB technical report TR 96-09, Berlin (1996)

Localized Communications of Data Parallel Programs on Multi-cluster Grid Systems[1]

Ching-Hsien Hsu, Tzu-Tai Lo, and Kun-Ming Yu

Department of Computer Science and Information Engineering,
Chung Hua University, Hsinchu, Taiwan 300, R.O.C
chh@chu.edu.tw

Abstract. The advent of widely interconnected computing resources introduces the technologies of grid computing. A typical grid system, the cluster grid, consists of several clusters located in multiple campuses that distributed globally over the Internet. Because of the Internet infrastructure of cluster grid, the communication overhead becomes as key factor to the performance of applications on cluster grid. In this paper, we present a processor reordering technique for the communication optimizations of data parallel programs on cluster grid. The alignment of data in parallel programs is considered as example to examine the proposed techniques. Effectiveness of the processor reordering technique is to reduce the inter-cluster communication overheads and to speedup the execution of parallel applications in the underlying distributed clusters. Our preliminary analysis and experimental results of the proposed method on mapping data to logical grid nodes show improvement of communication costs and conduce to better performance of parallel programs on different hierarchical grid of cluster systems.

1 Introduction

One of the virtues of high performance computing is to integrate massive computing resources for accomplishing large-scaled computation problems. The common point of these problems has enormous data to be processed. Due to cost-effective, clusters have been employed as a platform for high-performance and high-availability computing platform. In recent years, as the growth of Internet technologies, the grid computing emerging as a widely accepted paradigm for next-generation applications, such as data parallel problems in supercomputing, web-serving, commercial applications and grand challenge problems.

Differing from the traditional parallel computers, a grid system [7] integrates distributed computing resources to establish a virtual and high expandable parallel platform. Figure 1 shows the typical architecture of cluster grid. Each cluster is geographically located in different campuses and connected by software of computational grids through the Internet. In cluster grid, communications occurred when grid nodes exchange data with others via network to run job completion. These

[1] The work of this paper was supported by NCHC, National Center for High Performance Computing and NSC, National Science Council of Taiwan, under grant number NCHC-KING-010200 and NSC-93-2213-E-216-028, respectively.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 900 – 910, 2005.

communications are usually classified into two types, local and remote. If the two grid nodes belong to different clusters, the messaging should be accomplished through the Internet. We refer this kind of data transmission as *external communication*. If the two grid nodes in the same space domain, the communications take place within a cluster; we refer this kind of data transmission as *interior communication*. Intuitionally, the external communication is usually with higher communication latency than that of the interior communication since the data should be routed through numbers of layer-3 routers or higher-level network devices over the Internet. Therefore, to efficiently execute parallel applications on cluster grid, it is extremely critical to avoid large amount of external communications.

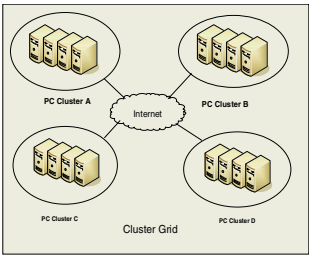


Fig. 1. The paradigm of cluster grid

In this paper, we consider the issue of minimizing external communications of data parallel program on cluster grid. We first employ the example of data alignments and realignments that provided in many data parallel-programming languages to examine the effective of the proposed data to logical processor mapping scheme. As researches discovered that many parallel applications require different access patterns to meet parallelism and data locality during program execution. This will involve a series of data transfers such as array redistribution. For example, a 2D-FFT pipeline involves communicating images with the same distribution repeatedly from one task to another. Consequently, the computing nodes might decompose local data set into sub-blocks uniformly and remapped these data blocks to designate processor group. From this phenomenon, we propose a processor-reordering scheme to reduce the volume of external communications of data parallel programs in cluster grid. The key idea is that of distributing data to grid/cluster nodes according to a mapping function at data distribution phase initially instead of in numerical-ascending order. We also evaluate the impact of the proposed techniques. The theoretical analysis and experimental results show improvement of volume of interior communications and conduce to better performance of data alignment in different hierarchical cluster grids.

The rest of this paper is organized as follows. Section 2 briefly surveys the related works. In section 3, we formulate the communication model of parallel data partitioning and re-alignment on cluster grid. Section 4 describes the processor-reordering scheme for communication localization. Section 5 reports the performance analysis and experimental results. Finally, we conclude our paper in section 6.

2 Related Work

Clusters have been widely used for solving grand challenge applications due to their good price-performance nature. With the growth of Internet technologies, the computational grids [4] become newly accepted paradigm for solving these applications. As the number of clusters increases within an enterprise and globally, there is the need for a software architecture that can integrate these resources into larger grid of clusters. Therefore, the goal of effectively utilizing the power of geographically distributed computing resources has been the subject of many research projects like Globus [6, 8] and Condor [9]. Frey *et al.* [9] also presented an agent-based resource management system that allowed users to control global resources. The system is combined with Condor and Globus, gave powerful job management capabilities is called Condor-G.

Recent work on computational grid has been broadly discussed on different aspects, such as security, fault tolerance, resource management [9, 2], job scheduling [17, 18, 19], and communication optimizations [20, 5, 16, 3]. For communication optimizations, Dawson *et al*. [5] and Zhu *et al*. [20] addressed the problems of optimizations of user-level communication patterns in local space domain for cluster-based parallel computing. Plaat *et al*. analyzed the behavior of different applications on wide-area multi-clusters [16, 3]. Similar researches were studied in the past years over traditional supercomputing architectures [12, 13]. For example, Guo *et al.* [11] eliminated node contention in communication phases and reduced communication steps with schedule table. Y. W. Lim *et al.* [15] presented an efficient algorithm for block-cyclic data realignments. Kalns and Ni [14] proposed the processor mapping technique to minimize the volume of communication data for runtime data re-alignments. Namely, the mapping technique minimizes the size of data that need to be transmitted between two algorithm phases. Lee *et al.* [10] proposed similar algorithms, the processor reordering, to reduce data communication cost. They also compared their effects upon various conditions of communication patterns.

The above researches give significant improvement of parallel applications on distributed memory multi-computers. However, most techniques only applicable for parallel programs running on local space domain, like single cluster or parallel machine. For a global grid of clusters, these techniques become inapplicable due to various factors of Internet hierarchical and its communication latency. In this paper, our emphasis is on dealing with the optimizations of communications for data parallel programs on cluster grid.

3 Preliminaries

3.1 Problem Formulation

The data parallel programming model has become a widely accepted paradigm for parallel programming on distributed memory multi-computers. To efficiently execute a parallel program, appropriate data distribution is critical for balancing the computational load. A typical function to decompose the data equally can be accomplished via the `BLOCK` distribution directive.

It has been shown that the data reference patterns of some parallel applications might be changed dynamically. As they evolve, a good mapping of data to logical processors must change adaptively in order to ensure good data locality and reduce inter-processor communication. For example, a global array could be equally allocated to a set of processors initially in BLOCK distribution manner. As the algorithm goes into another phase that requires to access fine-grain data patterns, each processor might divide its local data into sub-blocks locally and then distribute these sub-blocks to corresponding destination processors. Figure 2 shows an example of this scenario. In the initial distribution, the global array is evenly decomposed into nine data sets and distributed over processors that are selected from three clusters. In the target distribution, each node divides its local data into three sub-blocks evenly and distributes them to the same processor set in grid as in the initial distribution. Since these data blocks might be needed and located in different processors, consequently, efficient inter-processor communications become major subject to the performance of these applications.

Initial Distribution

Cluster-1			*Cluster-2*			*Cluster-3*		
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
A	B	C	D	E	F	G	H	I

Target Distribution

Cluster1			*Cluster2*			*Cluster3*			*Cluster1*			*Cluster2*			*Cluster3*			*Cluster1*			*Cluster2*			*Cluster3*		
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3	d_1	d_2	d_3	e_1	e_2	e_3	f_1	f_2	f_3	g_1	g_2	g_3	h_1	h_2	h_3	i_1	i_2	i_3

Fig. 2. Data distributions over cluster grid

To facilitate the presentation of the proposed approach, we assume that a global array is distributed over processors in BLOCK manner at the initiation. Each node is requested to partition its local block into K equally sub-blocks and distribute them over processors in the same way. The second assumption is that each cluster provides the same number of computers involved in the computation.

Definition 1: The above term K is defined as *partition factor*.

For instance, the partition factor of the example in Figure 2 is K=3. (Block A is divided into a_1, a_2, a_3, B is divided into b_1, b_2, b_3, etc.)

Definition 2: Given a cluster grid, C denotes the number of clusters in the grid; n_i is the number of processors selected from cluster i, where $1 \le i \le C$; P is the total number of processors in the cluster grid.

According to definition 2, we have $P = \sum_{i=1}^{C} n_i$. Figure 2 has three clusters, thus C = 3, where $\{P_0, P_1, P_2\} \in$ *Cluster* 1, $\{P_3, P_4, P_5\} \in$ *Cluster* 2 and $\{P_6, P_7, P_8\} \in$ *Cluster* 3, we also have $n_1 = n_2 = n_3 = 3$ and $P = 9$.

3.2 Communication Cost Model

Because the interface of interconnect switching networks in each cluster system might be different; to obtain accurate evaluation, the interior communication costs in clusters should be identified individually. We let T_i represents the time of two processors both reside in *Cluster-i* to transmit per unit data; m_i is the sum of volume of all interior messages in *Cluster-i*; for an external communication between cluster i and cluster j, T_{ij} is used to represent the time of processor p in cluster i and processor q in cluster j to transmit per unit data; similarly, m_{ij} is the sum of volume of all external messages between cluster i and cluster j. According to these declarations, we can have the following cost function,

$$T_{comm} = \sum_{i=1}^{C} T_i \times m_i + \sum_{i,j=1, i \neq j}^{C} (m_{ij} \times T_{ij}) \tag{1}$$

Due to various factors over Internet might cause communication delay; it is difficult to get accurate costs from the above function. As the need of a criterion for performance modeling, integrating the interior and external communications among all clusters into points is an alternative mechanism to get legitimate evaluation. Thus, we totted up the number of these two terms to represent the communication costs through the whole running phase for the following discussions. The volume of interior communications, denoted as $|I|$ and external communications, denoted as $|E|$ are defined as follows,

$$|I| = \sum_{i=1}^{C} I_i \tag{2}$$

$$|E| = \sum_{i,j=1, i \neq j}^{C} E_{ij} \tag{3}$$

Where I_i is the total number of interior communications within cluster i; E_{ij} is the total number of external communications between cluster i and cluster j.

4 Communication Localization

4.1 Motivating Example

Let us consider the example in Figure 2. In the target distribution, processor P_0 divides data block A into a_1, a_2, and a_3. Then, it distributes these three sub-blocks to processors P_0, P_1 and P_2, respectively. Since processors P_0, P_1 and P_2 belong to the same cluster with P_0; therefore, these are three interior communications. Similar situation on processor P_1 will generate three external communications; P_1 divides its local data block B into b_1, b_2, and b_3. It distributes these three sub-blocks to P_3, P_4 and P_5, respectively. However, as processor P_1 belongs to *Cluster* 1 while processors P_3, P_4 and P_5, belong to *Cluster* 2. Thus, this results three external communications. Figure 3 summarizes all messaging patterns of the example into a communication table. The

messages $\{a_1, a_2, a_3\}$, $\{e_1, e_2, e_3\}$ and $\{i_1, i_2, i_3\}$ are interior communications (the shadow blocks). All the others are external communications. Therefore, we have $|I| = 9$ and $|E| = 18$.

DP / SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_0	a_1	a_2	a_3						
P_1				b_1	b_2	b_3			
P_2							c_1	c_2	c_3
P_3	d_1	d_2	d_3						
P_4				e_1	e_2	e_3			
P_5							f_1	f_2	f_3
P_6	g_1	g_2	g_3						
P_7				h_1	h_2	h_3			
P_8							i_1	i_2	i_3
	Cluster-1			*Cluster*-2			*Cluster*-3		

Fig. 3. Communication table of data distribution over cluster grid

Figure 4 illustrates a bipartite representation to show the communications that given in the above table. In this graph, the dashed arrows and solid arrows indicate interior and external communications, respectively. Each arrow contains three communication links.

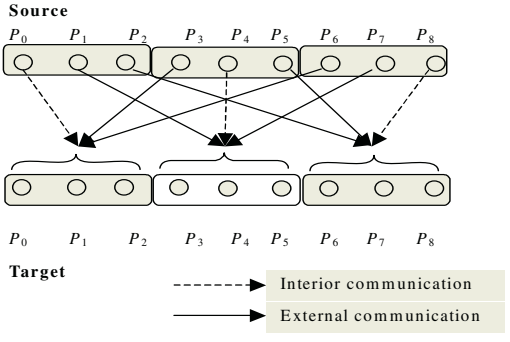


Fig. 4. Interior and external communications using bipartite representation

4.2 Processor Reordering Data Partitioning

The processor mapping techniques were used in several previous researches to minimize data transmission time of runtime array redistribution. In a cluster grid system, the similar concept can be applied. According to assumptions in section 3.1, we proposed the processor reordering technique and its mapping function that is applicable to data realignment on cluster grid. In order to localize the communication, the mapping function produces a reordered sequence of processors for grouping communications into local cluster. A reordering agent is used to accomplish this process. Figure 5 shows the concept of processor reordering technique for parallel data to logical processor mapping. The source data is partitioned and distributed to processors into initial distributions ($ID(P_X)$) according to the processor sequence derived from reordering agent, where X is the processor id and $0 \le X \le P$-1. To

accomplish the target distribution ($TD(P_{X'})$), the initial data is divided into K sub-blocks and realign with processors according to the new processors id X' that is also derived from the reordering agent. Given distribution factor K and processor grid (with variables C and n_i), for the case of $K=n_i$, the mapping function used in reordering agent is formulated as follows,

$$F(X) = X' = \lfloor X / C \rfloor + (X \bmod C) * K \tag{4}$$

We use the same example to demonstrate the above reordering scheme. Figure 6 shows the communication table of messages using new logical processor sequence. The initial distribution of source data is allocated by the sequence of processors' id, <P_0, P_3, P_6, P_1, P_4, P_7, P_2, P_5, P_8> which is derived from equation 4. To accomplish the target distribution, P_0 divides data block A into a_1, a_2, a_3 and distributes them to P_0, P_1 and P_2, respectively. These communications are interior. For P_3, the division of initial data also generates three interior communications; because P_3 divides its local data B into b_1, b_2, b_3 and distributes these three sub-blocks to P_3, P_4 and P_5, respectively; which are in the same cluster with P_3. Similarly, P_6 sends e_1, e_2 and e_3 to processors P_6, P_7 and P_8 and causes three interior communications. Eventually, there is no external communication incurred in this example in Figure 6.

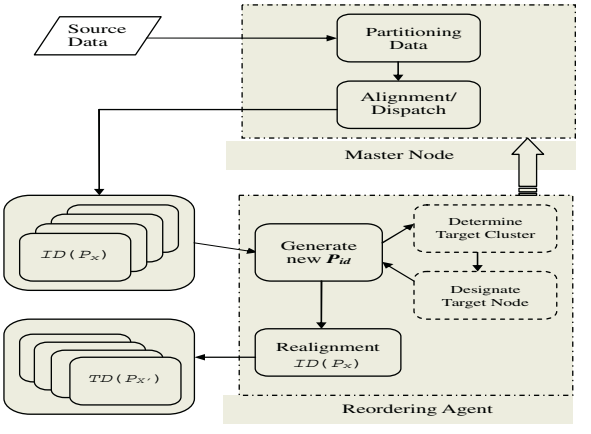


Fig. 5. The flow of data to logical processor mapping

DP / SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_0	a_1	a_2	a_3						
P_3				b_1	b_2	b_3			
P_6							c_1	c_2	c_3
P_1	d_1	d_2	d_3						
P_4				e_1	e_2	e_3			
P_7							f_1	f_2	f_3
P_2	g_1	g_2	g_3						
P_5				h_1	h_2	h_3			
P_8							i_1	i_2	i_3
	Cluster-1			Cluster-2			Cluster-3		

Fig. 6. Communication table with processor reordering

The bipartite representation of Figure 6's communication table is shown in Figure 7. All the communication arrows are in dashed lines. We totted up the communications, then have $| I | = 27$ and $| E | = 0$. The external communications are mostly eliminated.

5 Performance Analysis and Experimental Results

5.1 Performance Analysis

The effectiveness of processor reordering technique in different hierarchy of cluster grid can be evaluated in theoretical. This section presents the improvements of volume of interior communications for different number of clusters (C) and partition factors (K).

For the case consists of three clusters (C=3), Figure 8(a) shows that the processor reordering technique provides more interior communications than the method without processor reordering. For the case consists of four clusters (C=4), the values of K vary from 4 to 10. The processor reordering technique also provides more interior communications as shown in Figure 8(b). Note that Figures 8 and 9 report the theoretical results which will not be affected by the Internet traffic. In other words, Figure 8 is our theoretical predictions.

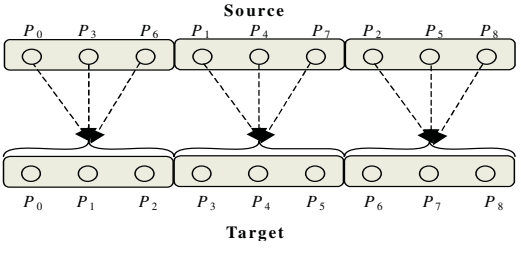


Fig. 7. Bipartite representation with processor reordering

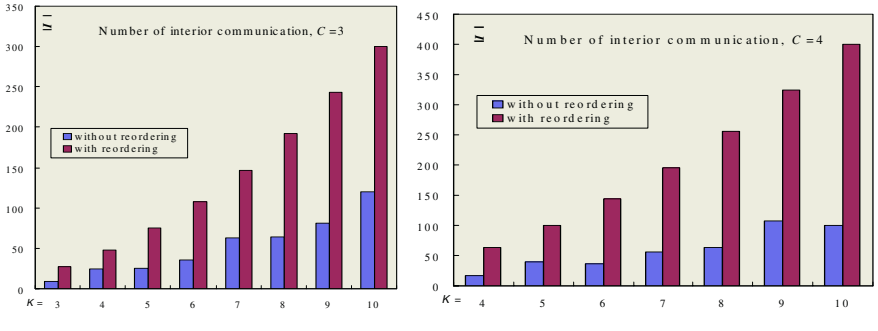


Fig. 8. The number of interior communications (a) C=3 (b) C=4

5.2 Simulation Settings and Experimental Results

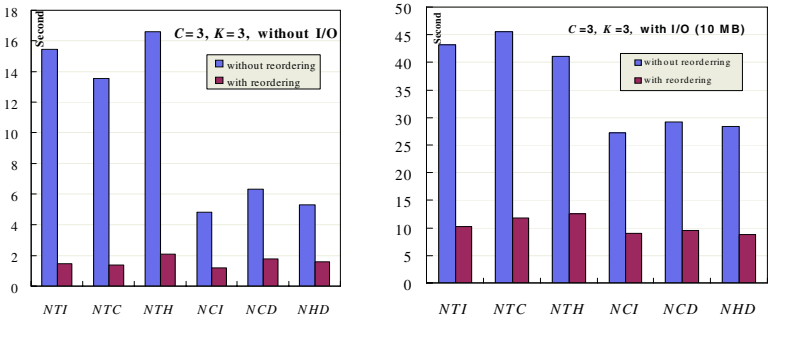
To evaluate the performance of the proposed technique, we have implemented the processor reordering method and tested on *Taiwan UniGrid* in which 8 campus clusters ware interconnected via Internet. Each cluster owns different number of computing nodes. The programs were written in the single program multiple data (*SPMD*) programming paradigm with C+MPI codes.

Figure 9 shows the execution time of the methods with and without processor reordering to perform data realignment when C=3 and K=3. Figure 9(a) gives the result of 1MB test data that without file system access (I/O). The result for 10MB test data that accessed via file system (I/O) is given in Figure 9(b). Different combinations of clusters denoted as *NTI*, *NTC*, *NTD*, etc. were tested. The composition of these labels is summarized in Table 1.

Table 1. Labels of different cluster grid

Label	*Cluster*-1	*Cluster*-2	*Cluster*-3	**Label**	*Cluster*-1	*Cluster*-2	*Cluster*-3
NTI	*NCHC*	*NTHU*	***IIS***	***NCI***	*NCHC*	***CHU***	***IIS***
NTC	*NCHC*	*NTHU*	***CHU***	***NCD***	*NCHC*	***CHU***	*NDHU*
NTH	*NCHC*	*NTHU*	***THU***	***NHD***	*NCHC*	***THU***	*NDHU*

In Figure 9(a), we observe that processor reordering technique outperforms the traditional method. In this experiment, our attention is on the presented efficiency of the processor reordering technique instead of on the execution time in different clusters. Compare to the results given in Figure 8, this experiment matches the theoretical predictions. It also satisfying reflects the efficiency of the processor reordering technique. Figure 9(b) presents the results with larger test data (10 MB) under the same cluster grid. Each node is requested to perform the data realignments through access file system (I/O). The improvement rates are lower than that in Figure 9(a). This is because both methods spend part of time to perform I/O; the ratio of communication cost becomes lower. Nonetheless, the reordering technique still presents considerable improvement.



(a) (b)

Fig. 9. Execution time of data realignments on cluster grid when $C = K = 3$

6 Conclusions and Future Works

In this paper, we have presented a processor reordering technique for localizing the communications of data parallel programs on cluster grid. Our preliminary analysis and experimental results of re-mapping data to logical grid nodes show improvement of volume of interior communications. The proposed techniques conduce to better performance of data parallel programs on different hierarchical grid of clusters systems. There are numbers of research issues remained in this paper. The current work of our study restricts conditions in solving the realignment problem. In the future, we intend to devote generalized mapping mechanisms for parallel data partitioning. We will also study realistic applications and analyze their performance on the UniGrid. Besides, the issues of larger grid system and analysis of network communication latency are also interesting and will be investigated.

References

1. Taiwan UniGrid, http://unigrid.nchc.org.tw
2. O. Beaumont, A. Legrand and Y. Robert, "Optimal algorithms for scheduling divisible workloads on heterogeneous systems," *Proceedings of the 12th IEEE Heterogeneous Computing Workshop,* 2003.
3. Henri E. Bal, Aske Plaat, Mirjam G. Bakker, Peter Dozy, and Rutger F.H. Hofman, "Optimizing Parallel Applications for Wide-Area Clusters," *Proceedings of the 12th International Parallel Processing Symposium IPPS'98*, pp 784-790, 1998.
4. J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta and K. Vahi, "The role of planning in grid computing," *Proceedings of ICAPS'03*, 2003.
5. J. Dawson and P. Strazdins, "Optimizing User-Level Communication Patterns on the Fujitsu AP3000," *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, pp. 105-111, 1999.
6. I. Foster, "Building an open Grid," *Proceedings of the second IEEE international symposium on Network Computing and Applications*, 2003.
7. I. Foster and C. K., "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, ISBN 1-55860-475-8, 1999.
8. I. Foster and C. Kessclman, "Globus: A metacomputing infrastructure toolkit," *Intl. J. Supercomputer Applications*, vol. 11, no. 2, pp. 115-128, 1997.
9. James Frey, Todd Tannenbaum, M. Livny, I. Foster and S. Tuccke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Journal of Cluster Computing*, vol. 5, pp. 237 – 246, 2002.
10. Saeri Lee, Hyun-Gyoo Yook, Mi-Soon Koo and Myong-Soon Park, "Processor reordering algorithms toward efficient GEN_BLOCK redistribution," *Proceedings of the 2001 ACM symposium on Applied computing*, 2001.
11. M. Guo and I. Nakata, "A Framework for Efficient Data Redistribution on Distributed Memory Multicomputers," *The Journal of Supercomputing*, vol.20, no.3, pp. 243-265, 2001.
12. Florin Isaila and Walter F. Tichy, "Mapping Functions and Data Redistribution for Parallel Files," *Proceedings of IPDPS 2002 Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications, Fort Lauderdale*, April 2002.

13. Jens Koonp and Eduard Mehofer, "Distribution assignment placement: Effective optimization of redistribution costs," *IEEE TPDS*, vol. 13, no. 6, June 2002.
14. E. T. Kalns and L. M. Ni, "Processor mapping techniques toward efficient data redistribution," *IEEE TPDS*, vol. 6, no. 12, pp. 1234-1247, 1995.
15. Y. W. Lim, P. B. Bhat and V. K. Parsanna, "Efficient algorithm for block-cyclic redistribution of arrays," *Algorithmica*, vol. 24, no. 3-4, pp. 298-330, 1999.
16. Aske Plaat, Henri E. Bal, and Rutger F.H. Hofman, "Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects," *Proceedings of the 5th IEEE High Performance Computer Architecture HPCA'99*, pp. 244-253, 1999.
17. Xiao Qin and Hong Jiang, "Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems," *Proceedings of the 30th ICPP,* Valencia, Spain, 2001.
18. S. Ranaweera and Dharma P. Agrawal, "Scheduling of Periodic Time Critical Applications for Pipelined Execution on Heterogeneous Systems," *Proceedings of the 30th ICPP,* Valencia, Spain, 2001.
19. D.P. Spooner, S.A. Jarvis, J. Caoy, S. Saini and G.R. Nudd, "Local Grid Scheduling Techniques using Performance Prediction," *IEE Proc. Computers and Digital Techniques*, 150(2): 87-96, 2003.
20. Ming Zhu, Wentong Cai and Bu-Sung Lee, **"**Key Message Algorithm: A Communication Optimization Algorithm in Cluster-Based Parallel Computing," *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, 1999.

VIRGO: Virtual Hierarchical Overlay Network for Scalable Grid Computing

Lican Huang

School of Computer Science, Cardiff University,
Cardiff CF24 3AA, UK
lican.huang@cs.cardiff.ac.uk

Abstract. This paper presents a virtual hierarchical overlay network– VIRGO for scalable Grid computing. This virtual hierarchical overlay network is self-organizing and decentralized, with an effective lookup protocol for routing messages. It contains an n-tuple replicated virtual tree structured network that differs from DHT-based P2P networks and random unstructured networks cached by least-recently used (LRU) and minimum difference (MinD)replacement strategies. It retains the partial-match query and robust aspects of unstructured P2P and the advantage of effective routing and guaranteed searching of structured P2P. The time complexity, space complexity and message-cost of VIRGO is $O(logN)$, where N is the total number of nodes in the network. Since LRU and MinD replacement strategies are used for caching route nodes, VIRGO is also a load-balanced network.

1 Introduction

Although client/server technology has been successful in many IT fields, such as WWW (HTTP), FTP and Web services, it has many shortcomings. It is unscalable, with a single point of failure and does not fully use resources at the network edge. P2P technology is now considered the next generation computing model. It is scalable, with efficient use of resources, and processing power at the edge of the network. There are two types of P2P technologies. Industrial P2P applications such as FreeNet[1] use an unstructured approach which routes nodes by a "flooding" algorithm. This has the advantages of partial-match querying and robustness, but may cause excess network traffic, and lacks guaranteed search. The structured approach is mostly based on Distributed_Hash_Table(DHT) technologies such as Pastry [2], CAN [3], and CHORD[4], and is effective in time complexity, but lack partial-match query capability and locality aspects, which limits their use in Grid computing.

At present, most Grid systems are designed according to the Client/Server (C/S) model other than the P2P model: Globus[5](GT3)and Web Services[6] are representative of the C/S approach. C/S technology has scalability problem. Iamnitchi et al.[7] use P2P technology in Grid computing, and JXTA[8] gives another P2P platform, but they do not yet achieve a complete solution.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 911–921, 2005.

In this paper, we present a virtual hierarchical overlay network using P2P technology for scalable Grid computing – VIRGO. VIRGO is a hybrid of structured P2P and unstructured P2P. It contains a virtual group tree overlay topology[9] and a random netlike topology cached by least-recently used (LRU)and minimum difference(MinD) replacement strategies. It is equally effective in routing messages as structured P2P, but retains the partial-match query and robustness aspects of unstructured P2P. The time complexity, space complexity and message-cost of the VIRGO lookup protocol is $O(logN)$, where N is the total number of nodes in the network. Due to the LRU and MinD replacement strategies for caching route nodes, VIRGO is also a load-balanced network. In addition to Grid computing, VIRGO can also be used in Distributed Domain Name Systems,distributed search engines, file sharing, and so on.

2 VIRGO-Virtual Hierarchical Overlay Network

The VIRGO virtual hierarchical overlay network is a hybrid of unstructured P2P and structured P2P technologies, and combines the advantages of both. In a virtual hierarchical overlay network, there is a core network that consists of stable servers that can access the Internet, and a surrounding circle of thin computers. Machines (called "clients") in the surrounding circle connect via a node (called the entrance node)to the core machines by Client/Server technology, whereas the core machines use P2P technology to interact among themselves. This strategy can make the network stable because when nodes join or leave only nodes in the core circle are required to be considered.

The virtual hierarchical overlay network consists of a prerequisite virtual group tree, which is similar to that described in the author's previous paper[9], and random connections cached by least-recently used and minimum distance replacement strategies. The virtual group tree is virtually hierarchical, with one root-layer, several middle-layers, and many leaf virtual groups. Among the nodes of the lower layer virtual groups, N-tuple gateway nodes in each group are chosen to form upper-layer groups, and from the nodes of these upper-layer groups to form upper-upper-layer groups in the same way, and this way is repeated until one root-layer group is formed. Random connections cached in a node's routing table are maintained by least-recently used (LRU)and minimum difference(MinD) replacement strategies. The LRU strategy gives a greater chance to frequently requested nodes. The MinD strategy gives a greater chance to nodes with differing top prefixes in domain names; this makes VIRGO a distributed network. With randomly cached connections, the netlike VIRGO avoids congestion in the root node of the virtual tree topology, but keeps the advantages of effective message routing in treelike networks.

In the virtual hierarchical overlay network, every Group has a unique group name, such as Science.Biology.Botony, that indicates the group's location in the

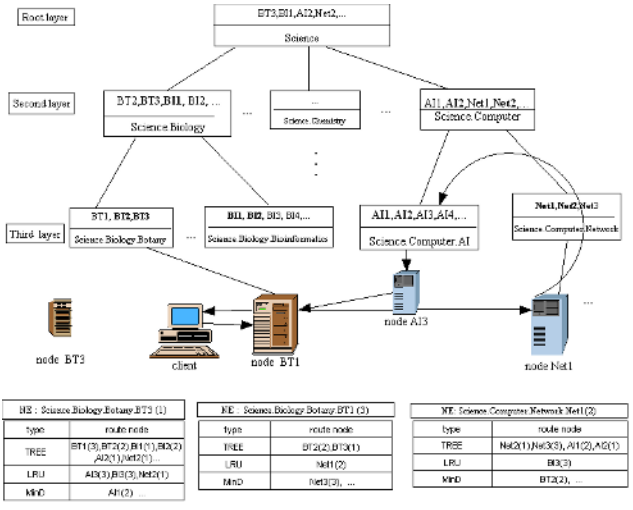


Fig. 1. Two_tuple Virtual Hierarchical Overlay Network

group tree. The members of a group are nodes with unique Domain Names such as Science.Biology.Botany.BT2(see Figure 1). The network in Figure 1 is a 2-tuple replicated virtual tree(The nodes with the same character in different layers are actually the same node). For example, BT3,BI1 in the root group come from the same second layer group Science.Biology. The routing table contains route nodes that are classified as 'TREE', 'LRU' and 'MinD' types. Every routing node has an ID and its gateway upmost layer, and exists within these groups from bottom layer to gateway upmost layer. BT1(3)means that the node is named as BT1 whose gateway upmost layer is in the third layer.

2.1 Definitions

Definition 1. ***User*** *(denoted by user) is the role which accesses the virtual hierarchical overlay networks.*

Definition 2. ***Client host*** *(denoted by cli) is an apparatus (such as desktop computer, PDA, mobile computer, etc), that is used by users to log into the virtual hierarchical overlay network.*

Definition 3. ***Node****(denoted by p) is a basic element in the virtual hierarchical overlay network. Every node has a Domain Name, as in DNS, and an Internet IP Address.* ***Gateway*** *(denoted by gn) is a node role which takes part in routing functions in several different layers of virtual groups. A gateway node is a node that is not only in a low-level group, but also is in an upper-level group.* ***Entrance*** *(denoted by ent) is an entrance point of a node for users to log into the virtual hierarchical overlay network.*

Definition 4. ***Domain name****(denoted by DomainName) is an hierarchical domain classification according to the purpose of the network. It is similar to DNS (for example, science.computer.network.Grid). Domain Name Length is the total layers of the Domain Name. In the above example, the length is 4. Every node has a unique Domain Name, and all nodes share a common first prefix (science in the above example).* ***Gateway uppermost layer****(denoted by GUL) is the uppermost virtual group layer that the gateway is in. The layers are ordered from root layer which is labelled as level 1.*

Definition 5. ***Virtual group****(denoted by VG) is formed virtually by the gateways nodes. VGα denotes virtual group with name α. The Group Name is part of the node's domain name, eg. in the above example, science, science.computer, science.computer.network are group names.*

Definition 6. ***N-tuple virtual group tree*** *(denoted by NVGT) is a hierarchical tree formed by virtual groups. Among the nodes of the lower layer virtual groups, N-tuple gateway nodes in each group are chosen to form upper-layer groups, and from the nodes of these upper-layer groups to form upper-upper-layer groups in the same way, and this way is repeated until a root-layer group is formed.*

3 Lookup Protocol

The lookup protocol for the virtual hierarchical overlay network is based on the strategy that among nodes in the routing table the node with the minimum distance to the destination node is chosen as the next hop on the route. The strategies of LRU and MinD are used for caching route nodes to solve computational and message congestion existing in the tree structure. Figure 1 shows the users using a client via entrance node BT1 (named Science.Biology.Botony.BT1) to request the entities of the destination node named Science.Computer.Network.AI3. The route path of this example is as follows. BT1 $->$ Net1 (because Net1 has the minimum distance to destination AI3 in BT1's routing table); then Net1$->$ AI1 (or AI2) ; then AI1(or AI2)$->$AI3.

To explain the lookup protocol in detail, we first give some brief definitions.

Definition 7. ***Node entity****(denoted by NE)is the node's Identification and status. It consists of the Domain Name, IP address, and GUL of the node.*

Definition 8. ***Item of route table****(denoted by IRT)is the record of* ***route table****(denoted by routetable). It consists of Node entity(NE), time of last use, and type. Type is classified as TREE, LRU and MinD. The node of TREE type is the node in the virtual tree network; a node of LRU type is cached by a least recently-used replacement strategy; and a node of MinD type is cached by a minimum difference replacement strategy. The items in the route table are ordered in alphabetical dictionary order by keyword of the node's Domain Name.*

Definition 9. ***Query message****(denoted by QUERYMESSAGE)is a request message by a client. It includes the domain name of requested node and requested entities such as keywords, etc.* ***Result message****(denoted by RESULTMESSAGE) contains the requested result and the node entity of the related destination node.* ***Join message*** *(denoted by JOINMESSAGE) is sent by a node applying to join the network.* ***Left message*** *(denoted by LEFTMESSAGE) is sent by a node to signal its departure from the network.* ***Failure message*** *(denoted by FAILUREMESSAGE) is sent by a node that is aware of a failure of communication with a gateway node.*

Some primitives and functions are listed in Table 1, and several details are as follows.

Table 1. Primitives and Functions

Names of Primitives and Functions and their Descriptions
sender.send (message, receiver) sender sends message to receiver
sender.send(message,receiver $\in Set$) sender sends message to all the receivers belong to a Set
macthNodeInRouteTable(DomainName) check if the node is in the route table
RouteTableUpdate(NE) update route table with node NE
RouteTableReplacement(NE, type) replace NE item of route table
RouteTableAdd(NE,type) add NE to route table
RouteTableDelete(NE,type) delete NE from route table
lookup_location(DomainName) find the node's location
LengthOfSamePrefix(DomainName, DomainName) length of shared prefixes between two nodes
LengthOfDomainName(DomainName) the length of DomainName
hopDistance2object(DomainName) the theoretical hops from the node to the destination node
selectRouteNodeFromRouteTable(DomainName) choose next hop node from route table
checkupQueryEntities(QUERYMESSAGE) check if the node has the entities queried
iscachesizeful(type) check if cached size in route table is full

The theoretical number of hops from a node to a destination node is calculated by the function hopDistance2object, whose details are described in the following pseudocode and Figure 2.

```
Function p.hopDistance2object(QUERYMESSAGE.DomainName)
  if LengthOfSamePrefix(p.DomainName, QUERYMESSAGE.DomainName) ==1
      return LengthOfDomainName(QUERYMESSAGE.DomainName)+p.GUL -2;
  elseif p.GUL ≤ LengthOfSamePrefix(p.DomainName, QUERYMESSAGE.DomainName)
      return LengthOfDomainName(QUERYMESSAGE.DomainName) -
          LengthOfSamePrefix(p.DomainName, QUERYMESSAGE.DomainName);
  else
      return LengthOfDomainName(QUERYMESSAGE.DomainName)+p.GUL -
          2*LengthOfSamePrefix(p.DomainName,QUERYMESSAGE.DomainName) ;
```

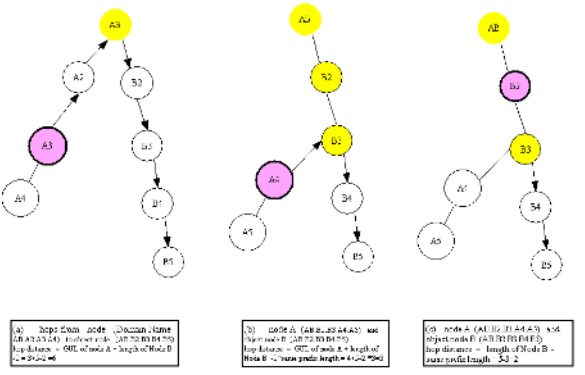


Fig. 2. Hop distance

The node with the minimum number of hops to the destination node is chosen as the next hop in the route. If there are more than one node with the same minimum number of hops in the route table, a random route node is chosen.

Function p.selectRouteNodeFromRouteTable(QUERYMESSAGE.DomainName)
gnSet =Minimum(p.hopDistance2object(gni ∈ $RouteTable, qmessage.DomainName$));
return routeP = random(gnSet);

Function *lookup_location* locates the destination node by the minimum hops

Function p.lookup_location(QUERYMESSAGE)
routeP = p.selectRouteNodeFromRouteTable(QUERYMESSAGE.DomainName);
if LengthOfSamePrefix(routeP.DomainName,QUERYMESSAGE.DomainName)==
LengthOfDomainName(QUERYMESSAGE.DomainName) {
RESULTMESSAGE= routeP.checkupQueryEntities(QUERYMESSAGE);
routeP.send(RESULTMESSAGE,ent); }
else {
p.send(QUERYMESSAGE,routeP);
if (message sending is successful) routeP.lookup_location(QUERYMESSAGE);
else { delete routeP from p.routetable; p.lookup_location(QUERYMESSAGE); } }

The lookup protocol queries the request entities and updates the cached LRU and MinD nodes in the route table. The algorithm used by the lookup protocol is as follows:

Step 1 cli.send (QUERYMESSAGE, ent)
Step 2 ent.lookup_location(QUERYMESSAGE.DomainName)
Step 3 ent.send(RESULTMESSAGE, cli);
Step 4 if ent.macthNodeInRouteTable(RESULTMESSAGE.ObjectNode.NE.DomainName)
ent.RouteTableUpdate(RESULTMESSAGE.ObjectNode.NE);

```
else {
  if ent.iscachesizeful(LRU)
    ent.RouteTableReplacement(RESULTMESSAGE.ObjectNode.NE, LRU);
    else ent.RouteTableAdd(RESULTMESSAGE.ObjectNode.NE,LRU);
    if ent.iscachesizeful(MinD)
      ent.RouteTableReplacement(RESULTMESSAGE.ObjectNode.NE,MinD);
    else ent.RouteTableAdd(RESULTMESSAGE.ObjectNode.NE,MinD);}
```

There exists a simple way to implement the RouteTableReplacement function for the MinD portion of the route table by sorting route nodes by the Domain names in alphabetical dictionary order.

4 Maintenance of Virtual Hierarchical Overlay Network

The maintenance of the virtual group tree, initialisation of the virtual tree network, the joining of new nodes, the departure of nodes, and the collapse of gateway nodes are all issues that need to be dealt with.

When a first node creates a new VIRGO network, it sets its own NE, which includes Domain Name, IP, GUL(value as 1) and type(value as TREE).

When a new node P_{join} joins, it first finds out the gateway node $P_{groupToJoin}$ which shares the maximum prefix length. The next steps are as follows.

1. P$_{join}$.$send(JOINMESSAGE, P_{groupToJoin})$
2. $P_{groupToJoin}.send(JOINMESSAGE, pi \in joinGroup)$;
3. $(pi \in joinGroup).send(pi.APPROVEMESSAGE, P_{join})$;
   $(pi \in joinGroup).RouteTableAdd(P_{join}.NE, TREE)$;
4. P_{join}.RouteTableAdd($pi \in joinGroup.NE$,TREE);
5. repeat step 2 to 4 in upper layer groups until replicated nodes no less than n-tuple or root group.

When a node P_{dpt} leaves the network, it will notify all the members of the virtual groups from the GUL to the bottom layer. These members will update their route tables. Each group will select a node to replace this left node's gateway role. The main steps in a node's departure from the network are as follows.

1. $P_{dpt}.send(LEFTMESSAGE, pi \in leftgroup)$;
2. $(pi \in leftgroup).RouteTableDelete(P_{dpt}.NE$,TREE);
3. choose $p_{replacenode}$ to replace the left node's role;
4. repeat step 1 and 3 in lower layer groups until to the bottom layer group of left node P$_{dpt}$.

Nodes may fail or depart without warning. Node P_{fail} is considered to have failed when node P_{notice} fails to communicate with node P_{fail}. Then node P_{notice} sends the FAILUREMESSAGE to all the members in the groups from P_{fail}'s GUL to the bottom layer. And every group chooses a node to replace P_{fail}'s gateway role.

1. $P_{notice}.send(FAILUREMESSAGE, pi \in failgroup)$;
2. $(pi \in failgroup).RouteTableDelete(P_{fail}.NE$,TREE);
3. choose $p_{replacenode}$ to replace P_{fail}'s role;
4. repeat step 1 and 3 in lower layer groups until to the bottom layer group of P_{fail}.

5 Analysis of Lookup Protocol

5.1 Connection

Theorem 1. *For an N-tuple virtual group tree, for any two nodes there exists a route path all the time with high probability.*

Proof: Suppose the time required to replace a failed or departing gateway node is T_r seconds, and the failure or departure frequency of a node in T_r is $F(p, T_r)$. The tree path length of the node to another node is *Length*, and P_f is the probability that a route path between two nodes fails to exist for all time. Due to the N_{tuple} replicated gateway nodes, we have:

$$P_f = F(p, T_r)^{N_{tuple}} * (Length - 1) \quad (1)$$

If T_r is 60 seconds, and each node fails on average 10 times per day, then $F(p, T_r)$ = 10*60/(24*60*60) =1/144. And if N-tuple is 6, and Length is 10, so P_f is $(1/144)^6 * 9 < 10^{-11}$.

This extremely small failure probability only affects performance because nodes can wait T_r time to route again, or the user may make the request once more.

5.2 Complexity

If the message is routed along the tree path, then every hop will reduce the distance from the destination node. Because the route table contains TREE portion, every hop reduces the distance from destination node by at least one hop. Therefore,

hops(a,b) and **message_cost** < length(a) + length(b)-1 (2)

time complexity = O(L) (3),

where L is the tree depth.

Because the gateway existing from root layer to bottom layer groups has the maximum route nodes in the route table, we have:

items of route table = L*N_tuple*nvg +Max_MinD + Max_LRU (4)

space complexity = O(L) (5),

where nvg is number of virtual groups, Max_MinD is the maximum number of MinD records in the route table, and Max_LRU is the maximum number of LRU records in the route table.

Suppose all leaf virtual groups have the same number of nodes, *nvg* and all non-leaf virtual groups have the same number of nodes, $nvg * n\_tuple$, then the number of layers L is $log_{nvg}N$, where N is the total size of the network. So,

time complexity and **message_cost** = O(log N) (6)

space complexity = O(log N) (7)

If the network has total number 10^8 nodes and 100 nodes per virtual group, then the routing hops to the destination is less than 8 ($2^* \ log_{100}10^8 = 8$).

Because VIRGO's lookup strategy is different to other DHT P2P systems, partial-query is possible and has the same time complexity by registering the contents in nodes according to the classification of domain names.

By using a LRU strategy for caching route table, routing to destination node can be partly reduced to just one hop.

RecordHitRate(RHR): The probability of caching a destination node in the LRU portion of route table. The number of LRU cached route table records in the route table is denoted by R_{LRU}. Ent_{user} is the number of users via an entrance node. $N(ent, user, t)$ is the number of nodes requested by a user via entrance during the time which covers a whole process from requesting information and getting the result back.

Due to the LRU replacement strategy, only recent nodes of interest during a short period need to be cached. Within this short period, $N(ent, user, t)$ is small. We have the following formula:

$$\mathrm{RHR}_{ent} = \mathrm{R}_{LRU}/(\bigcup^{Ent_{user}} N(ent, user, t)) \quad (8)$$

Supposing Ent_{user} is 100, $N(ent, user, t)$ is 10, and R_{LRU} 500, then RHR_{ent} $= 500/100 * 10 = 0.5$. This means that half of requests just need one hop.

Because formula (8) has no relation with the whole number of Grid system, a large scale system has the same Record Hit Rate. For mobile users, if we cache the nodes of interest in user's owner nodes, and after attaching entrance node, user first sends request to its owner, then we can obtain the same formula.

5.3 Load Balance

For P2P systems load balance is a very important issue. Because the load on the root layer node is the highest message load among all nodes in the network, we here only analyze the root node's message load.

We use two strategies to solve the load balance problem. The first is the LRU caching strategy as in the above section. The chance of routing a message through the root node is less than $1 - RHR_{ent}$. The second is the minimum difference (MinD) replacement strategy. To explain in detail,we first give some definitions.

MinDRecordHitRate($MinDRHR$): The probability of caching the shorter distance node in a node's MinD portion of the route table than the nodes in its TREE route table. The number of MinD cached records in the route table is denoted by R_{MinD}. $P_{msg}(user, t)$: The probability of the user's sending requested message at time t, which follows a Poisson distribution, is small. So, we have formula 9.

$$Root_{msg}(t) = P_{msg}(user, t) * N_{user} * (1 - RHR_{ent}) * (1 - MinDRHR)^{L-1} / n\_tuple(9),$$

where N_{User} is the total numer of users in the Grid system.

Supposing RHR_{ent} =0.2, MinDRHR=0.99, $NUser_{ent} = 10^{10}$, the number of nodes is 10^8,N_tuple is 6, nvg is 100,and $P_{msg}(user, t)$= 0.00001, then $L = log_{100}(10^8) = 4$. So, $Root_{msg}(t) = 0.00001 * 10^{10} * 0.8 * 0.01^3/6 = 0.013$.

How many of R_{MinD} are needed to make MinDRHR larger than 0.99 ? If there is route A in the route table which shares the first two prefixes with the destination node, then the next hop will bypass the nodes in the root layer and go directly to route A according to the selectRouteNodeFromRouteTable function and Figure 2. If the route table covers all of the first two prefixes in domain name space, then the request can route to a node in the cached MinD portion which has shorter distance than the nodes in the TREE portion of the route table. That is, the lookup protocol will bypass the root layer nodes for all destination nodes. So, the cache size of R_{MinD} only needs to be equal to the number of second layer virtual groups. Because the route nodes with different first two prefixes of domain names are randomly chosen from the huge number of nodes, there are no specific nodes with high traffic load. As Figure 1 shows, the MinD portions of the route tables in node Science.Biology.Botany.BT1 and Science.Biology.Botany.BT3 include Science.Computer.Network.Net3 and Science.Computer.AI.AI1, respectively. Therefore, BT1 and BT3 will locate the destination node AI3 by bypassing the root layer nodes. BT1 will directly route to Net3, whereas BT3 will route to AI1. So there are no specific nodes with high traffic load.

6 Conclusion

This paper presents a virtual hierarchical overlay network – VIRGO, that merges an n-tuple replicated virtual tree structured network and a random cached unstructured network by least-recently used (LRU)and minimum difference(MinD) replacement strategies. It is self-organizing and decentralized, with an effective lookup protocol for routing messages and a load balanced network. It retians the advantage of partial-match querying and the robustness of unstructured P2P networks, and the advantage of effective routing and guarantee search of structured P2P. The time complexity, space complexity, and message-cost for VIRGO are $O(logN)$ (N being the total number of nodes in the network). It can also be used in fields such as Distributed Domain Name System, distributed search engines, and file shares.

(The author thanks prof. David W. Walker for modifying the paper)

References

1. freenet,2004. http://freenet.sourceforge.net/
2. Rowstron, A. and Druschel, P., "Pastry:Scalable, distributed object location and routing for large-scale peer-to-pear systems", In Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001

3. Ratnasamy,S., Francis, P., Handley, K., Karp,R. and Shenker, S. , "A scalable content-addressable network", In Proceedings of ACM SIGCOMM 2001
4. Stoica,I. Morris,R., Karger,D., Kaashoek,F.M. and Balakrishnan, H., "Chord: a scalable peer-to-peer lookup service for internet applications", In Proceedings of ACM SIGCOMM2001
5. Foster,I. and Kesselman,C., "Globus: A Metacomputing Infrastructure Toolkit", *International Journal of Supercomputer Applications*, 11(2): 115-128, 1997
6. Web service, 2004. http://www.w3.org/TR/ws-arch/
7. Iamnitchi, A., and Foster, I., "On Fully Decentralized Resource Discovery in Grid Environments", International Workshop on Grid Computing 2001
8. jxta, 2004 http://www.jxta.org/
9. Huang Lican, Wu Zhaohui and Pan Yunhe, "Virtual and Dynamic Hierarchical Architecture for e-Science Grid", *International Journal of High Performance Computing Applications*, 17(3):329-347, 2003

A Monitoring Architecture for Control Grids

Alexandru Iosup[1], Nicolae Ţăpuş[1,⋆], and Stéphane Vialle[2,⋆⋆]

[1] Politehnica University of Bucharest,
Splaiul Independentei 313, Bucharest, Romania
{AIosup, NTapus}@cs.pub.ro,
[2] Supélec, 2 Rue Edouard Belin, 57070 Metz, France
Stephane.Vialle@metz.supelec.fr

Abstract. Monitoring systems are nowadays ubiquitous in complex environments, such as Grids. Their use is fundamental for performance evaluation, problem spotting, advanced debugging and per-use accounting. Building such systems raises challenging issues, like data gathering from Grid components, low intrusiveness, ease of use, adaptive data visualization, fault-tolerance and self-maintenance. This paper presents a new layered architecture, named Toytle, specifically designed to address these issues in the context of control Grids. All their components, from computing and network resources to complete physical processes with soft time constraints, can be monitored with Toytle. The architecture's layers, namely the distributed core, the hierarchical connections and the local monitors, have been designed to ensure scalability, high-speed sampling and efficient dealing with large data bursts. The future Toytle implementation will adapt existing tools and also create entirely new modules.

Keywords: control Grids, monitoring architecture.

1 Introduction

From the traditional computational and data Grids to the recent peer-to-peer and semantic Grids [2], there are many types of Grids and Grid applications that one can see nowadays. One of the most interesting examples is the emerging of control Grids, a special case of service grids [8], offering users the ability to control remote resources, including applications, computers and non-standard physical devices, in standardized ways. In general, control Grids serve as infrastructure for interdisciplinary projects, and are often used for applications adapted from the cluster and multicluster environments [16, 6]. As possible scenarios we mention mobile robotics applications [12] and intrusion detection sensors management, from the physical processes having soft time constraints field, and low-cost local collaborative environments [3], from government/academia.

⋆ This work was partially supported by the RoGrid national programme [15].
⋆⋆ This work was partially supported by the ACI-GRID programme.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 922–931, 2005.

One of the perennial activities for deploying and maintaining complex environments is monitoring. In particular, monitoring control Grids is a demanding task [10]. First, the nature of the resources can be truly heterogeneous, from computing and storing resources to mobile devices and even mobile robots. Second, given the applications often encountered in control Grids, the locality of available resources ranges from constrained (e.g. clusters) to mid-range (e.g. inter-country). Finally, all these resources must be put together in concerted applications, which must be monitored in full. Techniques for monitoring large cluster environments already exist [9] and methods for monitoring computational Grids have already been developed [18]. Yet, it has not been established whether any of these two, or a combination, can be used in the case of control Grids, especially with physical processes having soft time constraints.

This work addresses the issues related to monitoring control Grids, with a threefold contribution. First, it identifies the requirements of control Grids monitoring (see Sect.2). Second, it introduces an architecture that focuses on the identified requirements (see Sect.3). Third, it emphasizes a number of research issues related to the architecture and concerning distributed monitoring data aggregation, storage and retrieval (see Sect.4). We also put our results in the perspective of related work (see Sect.5) and present our conclusions (see Sect.6).

2 Monitoring Control Grids

Monitoring in Grid environments typically results in dealing with *Grid-awareness*, *scalability* and *standards-based communication* [4]. In addition, ensuring *low intrusiveness* on the monitored machines, *presenting* relevant *data* in meaningful ways, and guaranteeing degrees of *fault-tolerance*, should also be enforced.

Monitoring control Grids raises additional problems, like:

- gathering data from all Grid components (network, hosts and devices, middleware and applications),
- dealing with large "bursts" of information, especially for sensors fields, where data acquisition is performed simultaneously and periodically,
- performing high-speed sampling, as non-standard devices must be monitored in full, with many parameters being measured frequently,
- keeping the monitoring system lowly-intrusive, as the monitored resources may be sensitive to perturbations.

Even more, as control Grids are, in general, interdisciplinary projects, with most of the people involved having low technical skills, if some at all, the installation and management of monitoring tools are extremely important.

3 A Layered Architecture for Monitoring Control Grids

We have designed a new monitoring architecture, named Toytle, which addresses the main issues of monitoring control Grids and their respective applications (e.g. physical processes with soft time constraints).

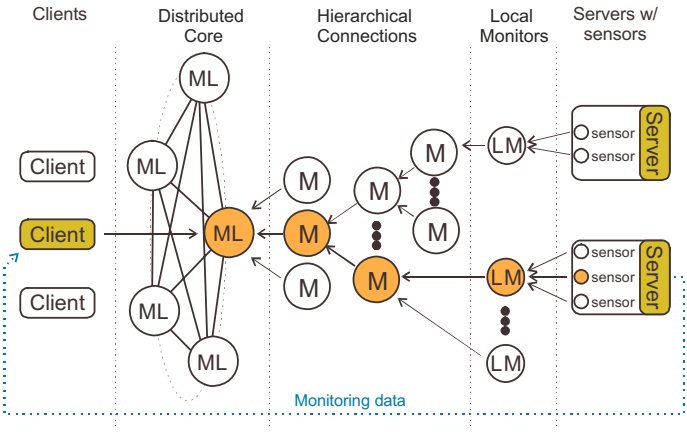


Fig. 1. The Toytle monitoring architecture components: *distributed core*, *hierarchical connections* and *local monitors*

3.1 Overview

The Toytle architecture is structured in three layers (see fig. 1):

1. **Distributed core layer** - a complex core with near cross-bar connections between nodes that provides high-level monitoring services to the user;
2. **Hierarchical connections layer** - a fairly light hierarchical structure that connects the local monitors to the distributed core, through data federation and piping;
3. **Local monitors layer** - a very light and lowly-intrusive set of monitors (i.e. not just sensors) that gather data from locally available sensors and perform some simple filtering on it.

The three layers cooperate for seamless monitoring services, in a hybrid, near *cross-bar core/N-level hierarchical connections* structure. By *near cross-bar* we mean that nodes inside the core have links to each other, resembling to some extent to a full cross-bar interconnection. By *N-level hierarchical connections* we mean the nodes that are not part of the core form a tree-like structure, with as many levels as needed.

It can be easily noticed that this architecture allows gathering data from all Grid layers data gathering. The hierarchical connections lead to a highly scalable approach; the data obtained from multiple large geographical areas (thus, multiple hierarchical trees) can be gathered in a distributed core, with ease of access added together with some degree of fault tolerance. The local monitors are *light-weigth* and correspond to the requirements of control Grids.

3.2 The Distributed Core Layer

The distributed monitoring core deals with issues like data gathering, storage, replication and visualization. Having a horizontal structure (see fig.2), with nodes

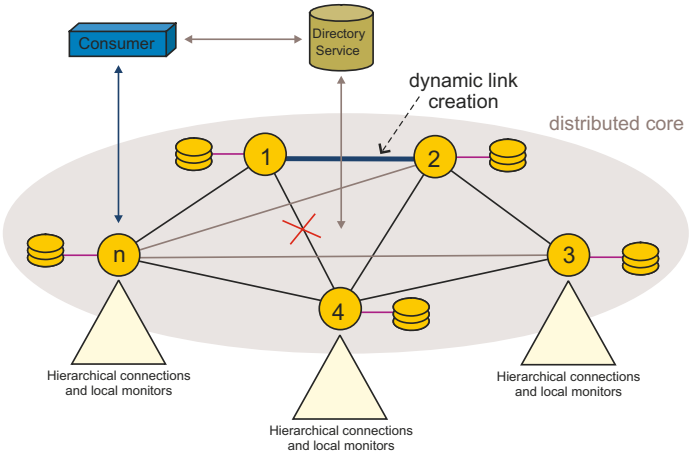


Fig. 2. The Toytle architecture distributed core layer. The nodes within the core can maintain permanent contact (thick connection lines between nodes) or just be able to communicate with one another (thin connection lines between nodes)

distributed according to geographical or problem-wise needs, this module tackles issues like adaptive system balancing, fault-tolerance, self-maintenance, Grid data access and user control.

The monitoring core nodes are able to *transparently balance* the monitoring system use. First, when a user tries to connect to a core node, given that the contacted node is too crowded, it will be automatically relocated to another core node. Second, data summarization should be performed by the least charged core node that has access to that data. For both these tasks, the core should also be self-monitoring.

Fault-tolerance is ensured through data replication. Data storage is performed per-node or per-system in one or more repositories. Recent data is stored in cache-like memory buffers, for faster access. Recent data replication is performed transparently for the user between core nodes. Data visualization allows for both resource- and application- oriented views. The monitoring core is self-maintained, so that node insertion and failure are dealt with transparently to the user. A link failure between two core nodes (see the cross in fig.2) can be transparently replaced by another link creation (see the link between nodes 1 and 2 in fig.2).

Also, in the event that the core node that was maintaining the connection with a lower-layer node (a root monitor from the hierarchical connections layer) falls, the core should transparently relocate the now broken connection to another core node. In other words, as long as the core contains at least one node, data is available to the users, and as long as several nodes are active at the core level, data federation is optimized for speed issues.

User control of the monitoring system is done through the specification of monitoring levels for all four categories of (control) Grids monitoring data: host, network, middleware and user applications. Every characteristic monitored by

the monitoring system has a type and a monitoring level associated with it. Once the monitoring level for the characteristic's type has been surpassed, that characteristic becomes monitored and its data starts reaching the core, thus being available to the user. The higher the monitoring level is, the larger the size of data gathered and summarized at the core level gets.

Since the number of core nodes typically does not exceed $1-2\%$ of the total number of resources in the monitored Grids, the near cross-bar interconnection at this level does not raise *scalability* issues. The amount of extra data space required for replication depends on the replication strategy (see Sect. 4, *data consistency and data replication* paragraph) and on the filtering options employed by the user. The low number of core nodes makes this extra-resource usage affordable. The fact that the core nodes are not light does not interfere with the *low intrusiveness* requirement of control Grids, as the core nodes do not actually run on the monitored system's machines.

Finally, in order to comply with the GGF GMA (e.g. consumer / producer / directory architecture), information about the monitoring core's data type and actual data location can be obtained from a monitoring directory. Also, standard data representations, both with size reduction (e.g. XDR [5]) and scheme self-definition (e.g. XML) targets, are offered.

3.3 The Hierarchical Connections

The *hierarchical connections layer* of the monitoring system performs data gathering and summarizing. It is also responsible for ensuring hierarchical monitoring connections, while keeping the monitored system's load at a minimum, and for transmitting changes in monitoring levels to the local monitors.

The vertical connection structure (see fig.3) allows for efficient data-summarization, aiming to reduce the load of the monitoring system. Each monitoring hierarchy top node is linked with one or more monitoring core nodes. As long as at least one of its nodes is active, the hierarchical connections layer continues to perform as a link between the core and the local monitors. This fault tolerance target is designed to be performed transparently to the user.

Temporary data recovery, in the context of low intrusiveness, is offered through backups on fixed-size (e.g. using a round-robin storage strategy) databases. As for the core layer, standard data representations, both with size reduction and scheme self-definition targets, are offered at this layer.

3.4 The Local Monitors

The *local monitors layer* performs all the data acquisition and simple on-the-fly filtering. At this level, locally-close monitors are connected in horizontal structures (see fig.4), with important benefits like node insertion and failure being dealt with transparently to the user. Data from all of the Grid components, network, hosts, middleware and applications, can be acquired at this level.

Fault-tolerance is ensured through data advertisements. Each local monitor advertises its data and sends it to other local monitors, so that exact replicas of that data can be temporarily found on other places than the origin, at any time.

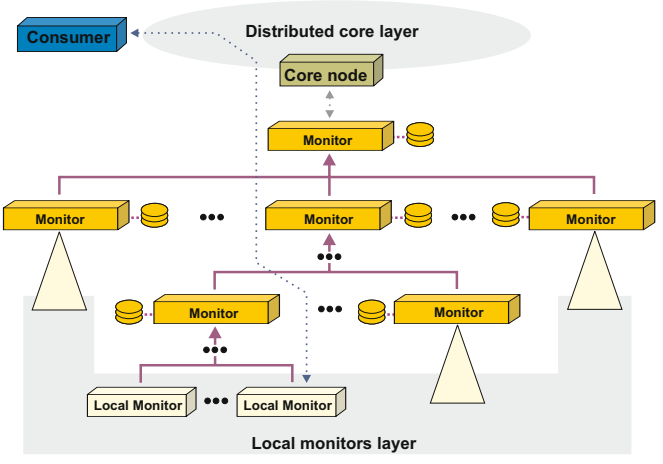


Fig. 3. The Toytle architecture hierarchical connections layer. Monitors form an N-level hierarchical structure. Monitors can store data in local databases. Note that consumers cannot directly access nodes at this level

Simple filtering should be performed at this level. By *simple* we mean filtering characteristics with thresholds and data correlation. Filtering with thresholds is done by setting certain limits, called *thresholds*, such that if a characteristic goes over the threshold, it gets immediately reported. Filtering by data correlation means not only defining thresholds for any given characteristic, but also combinations of thresholds for combinations of characteristics. For example, the condition could be that CPU load is less than 50% *and* available RAM is under 10% of the total RAM. Filtering by any of the these means should be done on all Grid data targets: network, hosts, middleware and especially user applications.

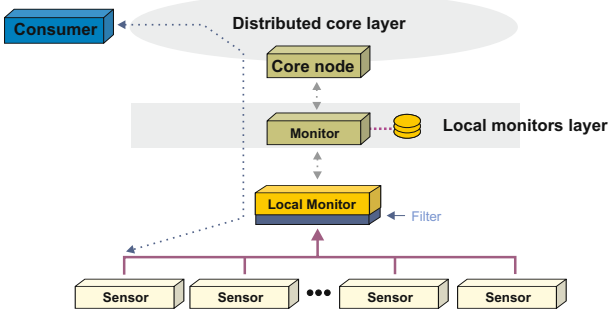


Fig. 4. The Toytle architecture local monitors layer. Local monitors and sensors form a 1-level hierarchical structure. Note that consumers cannot directly access nodes at this level

Two different *data pruning* methods are used at this level: *derived data pruning* and *monitoring level pruning*. Derived data pruning means that data that can be derived from other available data is not transmitted trough the hierarchical connections layer. Pruning according to a specified monitoring level means that each characteristic has a verbose level and must be acquired once the monitoring level reaches it. In any case, the low-intrusiveness requirement is that generated traffic for data transfers should be less than 1% of the theoretical bandwidth of the link (e.g. 0.1 $Mbps$ for a 10 $Mbps$ Ethernet connection, 1 $Mbps$ for a 100 $Mbps$ Fast Ethernet connection, and 10 $Mbps$ for a 1000 $Mbps$ Gigabit Ethernet connection).

The sampling rate of this level must be of at least 10 $samples/second$. This targeted sample rate has been derived from the normal local area network round-trip time (the time needed for a single packet to get from a source to a destination and back), which is nowadays just below 0.1s. A recommended sample rate should be, however, around 20 $samples/second$ (twice the minimum rate).

This layer also sends data to the upper layer in standard data formats, like XDR and XML. We emphasize that a compact and portable data transfer standard, like XDR, must be used at this level, because of the sheer size of the monitoring data.

3.5 Towards Implementing the Monitoring Architecture

Our first attempts of implementing the monitoring architecture focused on finding existing open-source monitoring tools, which may be minimally modified in order to integrate within Toytle. This approach allows us to focus on the novel aspects of the Toytle architecture, instead of dealing with already solved issues.

SNMP v2(c)/3 [13] is a flat structure standard for monitoring, which can deal with a large number of network device types, thus being an obvious candidate for the local monitors layer. Unfortunately, our tests on NET-SNMP, a renowned SNMP package, indicate that both its response latency and its generated traffic are too high for our mission.

Ganglia [9] has a hierarchical, tree-like, structure and is well-suited for high-speed data aggregation. The mechanisms employed by Ganglia for fault-tolerance, namely auto-managed multicast channel and periodic *heart-beat* signal, cover very well our requirements in these areas, at the local monitors level. Our tests show that Ganglia should be improved with adequate user data definition, a compressed data exchange protocol within the tree, fault-tolerance, and enhanced data filtering. Therefore, Ganglia is a very good candidate for future integration at the hierarchical connections and local monitors layers.

4 Research Issues

In this section we identify a number of open research issues concerning the distributed monitoring data aggregation, storage and retrieval, all in the context of the Toytle monitoring architecture.

Core Connectivity Factor Impact. At the extremes, the core can have either ring or full-mesh topologies. In-between, peer-to-peer connectivity and data lookup strategies may be applied [14], to ensure certain performance guarantees. Therefore, the performance of the system with various core topologies remains to be established.

Data Consistency and Data Replication. In order to ensure geographical and administrative scalability, several data consistency models can be used for the core layer. Beside the traditional strict consistency, the *conit*-based continuous consistency model [20] seems to be a promising alternative. Further issues regarding replica granularity, replica placement, and user characteristics must be studied for the devise of an adequate data replication strategy.

Data Query Language. The information stored by the monitoring service may be retrieved using targeted queries. It is debatable whether the large amounts of data, with *bursty* behavior, should be accessible through a standard language, like SQL, or through a customized one.

Adaptive Caching. The low/middle layers of the proposed monitoring architecture deal with repeated data transfers. Is storing data within a fixed-size (round-robin) table enough, or should there be an option for dynamically-sized disk (scratch) tables? Should there be a memory caching before writing to disk? We believe that such questions must find their answers in the context of real-world control Grids applications.

5 Related Work

There has been extensive work within the field of monitoring parallel and distributed systems. Recently, much interest has been put into monitoring Grid systems. A good survey on Grid monitoring can be found in [4]. However, we claim that nowadays monitoring systems cannot address in full monitoring control Grids and applications running in such environments.

Two monitoring systems are very close to our view on the monitoring core. First, Astrolabe [17] addresses peer-to-peer distributed information management, with special focus on scalability, but uses full-scale *gossip*ing (everyone gets to know everything), which may render the system impractical for real-time data aggregation. Second, MonALISA [11] has a flat structure and is well suited for data visualization and high-speed sampling, but lacks core connections and is not open-source, both of which issues are vital for our purposes.

Systems like DataGrid's GRM/Prove [7] and NetLogger[1] lack a lowly intrusive multi-level monitoring hierarchy. The NWS [19] system cannot be used with "bursty" applications and is not suitable for fast changing data sets. Systems like R-GMA[2] and MDS[3] are too resource-demanding and/or slow-paced to be

[1] NetLogger, http://www-didc.lbl.gov/NetLogger
[2] R-GMA, http://www.r-gma.org
[3] Monitoring and Discovery Service, http://www.globus.org/mds/

effective for the case of control Grids. CrossGrid's OCM-G [1] leaves up to the user to collect the data and share it between the nodes that request it, thus being impractical for control Grids. The GrADS Autopilot [18] toolkit is targeted at automatic performance monitoring and applications steering in the context of computational Grids, thus having a different focus than ours.

6 Conclusions and Future Work

This paper has presented a layered monitoring architecture, named Toytle, specifically designed for addressing the special requirements of control Grids. The architecture is a hybrid near cross-bar core/N-level hierarchical connections monitoring architecture, that aims at satisfying the most important requirements of control Grids. High speed data gathering and federation, scalability, program lightness (in terms of monitored system's resource consumption), low intrusiveness (in terms of monitored system's performance degradation) and easy deployment are targeted by this architecture. The architecture's three layers, namely the distributed core, the hierarchical connections and the local monitors, ensure a good degree of modularity, while still offering the advantages of architectural compactness.

For the future, we plan to develop new modules for the core layer, as well as adapting Ganglia to the hierarchical connections and the local monitors layers. This will lead to a complete implementation of Toytle, and will give us a good platform for addressing the research issues presented in Sect.4. Also, testing the complete implementation of Toytle against an existing Grid mobile robotics application will help identifying possible improvements to this monitoring architecture.

References

[1] B. Balis, M. Bubak, W. Furnika, T. Szepieniec, R. Wissmueller, and M. Radecki. Monitoring Grid applications with Grid-enabled OMIS monitor. In *Proceedings of the First European Grids Conference, AxGrid 2003, Santiago de Compostela, Spain*, pages 230–239. Springer Verlag, February 2003.

[2] F. Berman, A. Hey, and G. Fox. *Grid Computing: Making The Global Infrastructure a Reality*. Wiley Publishing House, 2003. ISBN: 0-470-85319-0.

[3] G. Fox. Experience with distance education 1998-2003. Collection of resc., http://grids.ucs.indiana.edu/ptliupages/publications/disted/.

[4] M. Gerndt, R. Wismueller, and Z. Balaton et al. Performance tools for the grid: State of the art and future. Technical report, APART WP3, 2004.

[5] Network Working Group. eXternal Data Representation (XDR), August 1995. IETF RFC 1832.

[6] A. Iosup and S. Vialle. Mobile robot navigation and self-localization system: Parallel and distributed experiments. In *The Dagstuhl Workshop on Plan-Based Control of Robotic Agents, Dagstuhl, Germany*, June 2003.

[7] P. Kacsuk. Parallel program development and execution in the grid. In *IEEE International. PARELEC'02, Warszaw, Poland*, pages 131–141, September 2002.

[8] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of Grid resource management systems for distributed computing. *Software Practice and Experience*, 32(2):135–164, February 2002.

[9] M. Massie, B. Chun, and D. Culler. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, July 2004.

[10] Zs. Nemeth, G. Gombas, and Z. Balaton. Performance evaluation on Grids: Directions, issues, and open problems. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04), A Coruna, Spain*, pages 290–297. IEEE Computer Society Press, February 2004.

[11] H.B. Newman, I.C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. MonALISA: A distributed monitoring service architecture. In *CHEP 2003, La Jola, California*, March 2003.

[12] F. Sabatier, A. De Vivo, and S. Vialle. Grid programming for distributed remote robot control. *International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004)*, June 2004. (to appear).

[13] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2 (3rd Ed.)*. Wiley Publishing House, 1998. ISBN: 0-201-48534-6.

[14] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, and M.F. Kaashoek. CHORD: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11:17–32, 2003.

[15] N. Tapus, V. Cristea, M. Burcea, and V. Staicu. RoGrid towards a Romanian computational Grid. In *Proceedings of the 14th International Conference on Control Systems and Computer Science (CSCS14), Romania*, July 2004.

[16] N. Tapus, E. Slusanschi, and T. Popescu. Distributed rendering engine. In *Proc. of the NATO Advanced Research Workshop on Adv. Environments, Tools, and Applications for Cluster Computing*, pages 207–215. Springer-Verlag, 2002.

[17] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[18] J.S. Vetter and D.A. Reed. Real-time performance monitoring, adaptive control, and interctive steering of computational grids. *The International Journal of High-Performance Computing Applications*, 14:357–366, Winter 2000.

[19] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, 2003.

[20] H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems*, 20(3):239–282, 2002.

Mobile-to-Grid Middleware: Bridging the Gap Between Mobile and Grid Environments

Hassan Jameel[1], Umar Kalim[1], Ali Sajjad[1],
Sungyoung Lee[1], and Taewoong Jeon[2]

[1] Department of Computer Engineering, Kyung Hee University,
1, Sochen-ri, Giheung-eup, Yongin-si, Gyeonggi-do, 449-701, South Korea
{hassan, umar, ali, sylee}@oslab.khu.ac.kr
[2] Department of Computer & Information Science, Korea University, Korea
jeon@selab.korea.ac.kr

Abstract. Currently, access to Grid services is limited to resourceful devices such as desktop PCs but most mobile devices (with wireless network connections) cannot access the Grid network directly because of their resource limitations. Yet, extending the potential of the Grid to a wider audience promises increase in flexible usage and productivity. In this paper we present a middleware architecture[1] that addresses the issues of job delegation to a Grid service, support for offline processing, secure communication, interaction with heterogeneous mobile devices and presentation of results formatted in accordance with the device specification. This is achieved by outsourcing the resource intensive tasks from the mobile device to the middleware. We also demonstrate through formal modeling using Petri nets that the addition of such a middleware causes minimum overhead and the benefits obtained outweigh this overhead.

1 Introduction

Grid [18] computing permits participating entities connected via networks to dynamically share their resources. Extending this potential of the Grid to a wider audience, promises increase in flexibility and productivity, particularly for the users of mobile devices who are the prospective consumers of this technology.

Consider a teacher who wants to augment his lecture with a heavy simulation test. He uses his PDA to access a Grid service and submit the request. The service after executing the request compiles the results which are then distributed to the mobile devices of the registered students of that course. Similarly a doctor on the way to see his patient, requests a Grid medical service to analyze the MRI or CT scans of the patient from his mobile device. By the time he meets his patient; the results would be compiled and presented on his mobile device to facilitate the treatment.

[1] This research work has been supported in part by the Korea Ministry of Information and Communications' ITRC program in joint collaboration with Information and Communications University, Korea.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 932–941, 2005.

The clients that interact with the Grid middleware to accomplish a task are required to use client end libraries. These libraries are relatively resource intensive considering the limitations of mobile devices. Conceiving a distributed system that uses these libraries directly will not be a practical mobile system because of the resource demands.

Moreover, most of the conventional distributed applications are developed with the assumption that the end-systems possess sufficient resources for the task at hand and the communication infrastructure is reliable. For the same reason, the middleware technologies for such distributed systems usually deal with issues such as heterogeneity and distribution (hence allowing the developer to focus his efforts on the functionality rather than the distribution).

The issues that primarily affect the design of a middleware for mobile systems are: *mobile devices*, *network connection*, and *mobility*. Firstly, due to the tremendous progress in development of mobile devices, a wide variety of devices are available which vary from one to another in terms of resource availability. Secondly, in mobile systems, network connections generally have limited bandwidth, high error rate and frequent disconnections due to power limitations, available communication spectrum and user mobility. Lastly, mobile clients usually interact with various networks, services, and security policies as they move from one place to another.

Considering the assumptions and characteristics of conventional middleware technologies it is quite evident that they are not designed to support mobile systems adequately. Instead, they aim at a static execution platform (where the host location is fixed) and the network bandwidth does not vary. Hence, given the highly variable computing environment of mobile systems, it is mandatory that modern middleware systems are designed that can support the requirements of mobile systems such as *dynamic reconfiguration and asynchronous communication*.

In this paper:

- We present an architecture for a middleware (Section 2) enabling heterogeneous mobile devices access to Grid services by providing support for delegation of jobs to the Grid, secure communication between the client and the Grid, off-line processing, adaptation to network connectivity issues and presentation of results in a form that is in keeping with the resources available at the client device.
- We demonstrate (Section 3) that the addition of such a middleware causes minimum overhead and the benefits obtained by it outweigh this overhead.

2 Architecture Details

The middleware service is exposed as a web service to the client applications. The components of the middleware service (as shown in Figure 1) are discussed succinctly as follows:

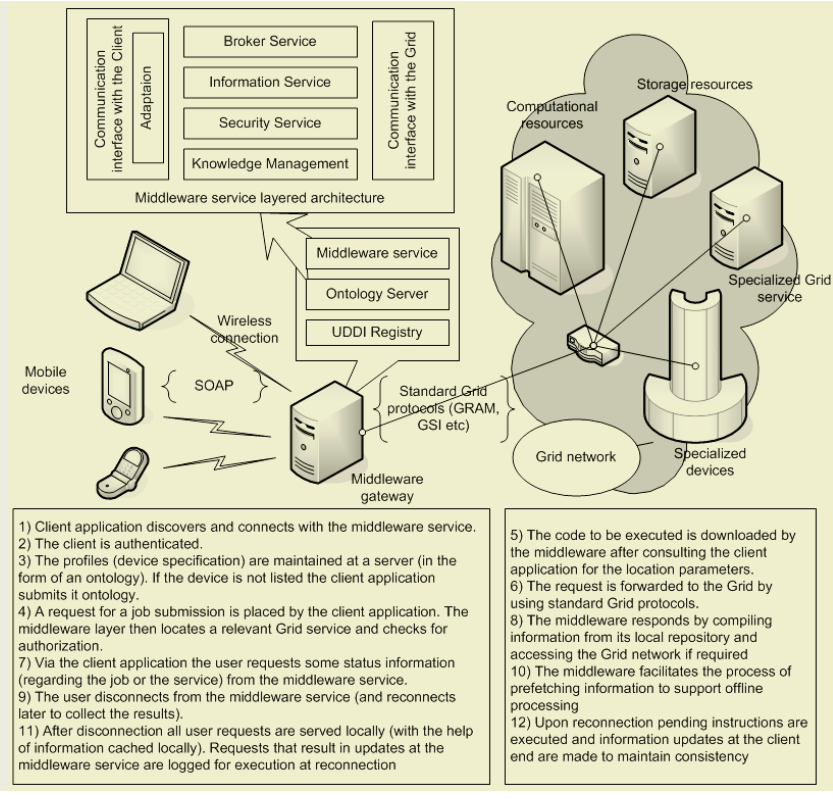


Fig. 1. Deployment model and the architecture

2.1 Discovery Service

The discovery of the middleware by mobile devices is managed by employing a UDDI registry [6], [7]. Once the middleware service is deployed and registered, other applications/devices would be able to discover and invoke it.

2.2 Communication Interface with the Client Application

The interface advertised to the client application is the communication layer between the mobile device and the middleware. This layer enables the middleware to operate as a web service and communicate via the SOAP framework [8].

Adaptation to Disconnected Operations. The advertisement of the mobile-to-Grid middleware as a web service permits the development of the architecture in a manner that does not make it mandatory for the client application to remain connected to the middleware at all times while the request is being served.

We focus on providing software support for offline processing at the client device. For this we consider two kinds of disconnections; intentional disconnection,

where the user decides to discontinue the wireless connection and unintentional disconnection, which might occur due to variation in bandwidth, noise, lack of power etc. This is made possible by pre-fetching information or meta-data only from the middleware service. This facilitates in locally serving the client application at the device. However, requests that would result in updates at the middleware service are logged (so that they may be executed upon reconnection).

To establish the mode of operation for the client application, a connection monitor is used to determine the network bandwidth and consequently the connection state (connected or disconnected). Moreover, during execution, checkpoints are maintained at the client and the middleware in order to optimize reintegration after disconnection.

2.3 Communication Interface with the Grid

The communication interface with the Grid provides access to the Grid services by creating wrappers for the API advertised by the Grid. These wrappers include standard Grid protocols such as GRAM [9], MDS [10], GSI [11] etc which are mandatory for any client application trying to communicate with the Grid services. This enables the middleware to communicate with the Grid, in order to accomplish the job assigned by the client.

2.4 Broker Service

The broker service deals with initiating the job request and steering it on behalf of the client application. Firstly the client application places a request for a job submission. After determining the availability of the Grid service and authorization of the client, the middleware downloads the code (from the mobile device or from a location specified by the client e.g. an FTP/web server). Once the code is available, the broker service communicates with Grid's GRAM service to delegate the job.

A status monitor service (a subset of the broker service) interacts with GRAM to determine the status of the job. The status monitor service then communicates with the Knowledge Management module to store the results. The mobile client may reconnect and ask for the (intermediate/final) results of its job from the status monitor service.

2.5 Knowledge Management

The knowledge management layer of the system is used to manage the relevant information regarding both the client and Grid applications and services. The main function of this layer is to connect the client and Grid seamlessly as well as to introduce the capability of taking intelligent decisions such as downscaling the results according to device profile, based on the information available to the system.

2.6 Information Service

This module interacts with the wrapper of the GLOBUS toolkit's API for information services (MDS [10]). It facilitates the client application by managing the process of determining which services and resources are available in the Grid and also monitors resources such as CPU load, free memory etc.

2.7 Security

The Grid Security Infrastructure is based on public key scheme mainly deployed using the RSA algorithm [12]. However key sizes in the RSA scheme are large and thus computationally heavy on handheld devices such as PDA's, mobile phone's, smart phones etc [13]. We employ the Web Services Security Model [14] to provide secure mobile access to the Grid. This web services model supports multiple cryptographic technologies.

The Elliptic Curve Cryptography (ECC) based public key scheme can be used in conjunction with Advanced Encryption Standard(AES) for mobile access to Grid which provide the same level of security as RSA and yet the key sizes are a lot smaller [13].

Communication between the user and middleware is based on security policies specified in the user profile. According to this policy different levels of security can be used e.g. some users might just require authentication, and need not want privacy or integrity of messages.

3 Petri Net Model of the System and Its Analysis

In this section we model the interaction between the mobile client and the mobile grid middleware service. Our goal is to estimate the delay caused by the communication between the client and middleware service as well as the additional processing done by it. This delay should be within acceptable limits so that the mobile client user is not at a disadvantage as compared to a normal Grid user as far as time is concerned. We use the time to completion of the whole process as an index of performance of our middleware communication architecture. We keep the time taken by Grid processing constant as our results will be bench marked against it. The communication is modeled by using non-Markovian Stochastic Petri nets [2] [3]. We follow an approach similar to the work done by Antonio Puliafito et al [1].

To make the Petri Net model in Figure 2, we modeled the following sequence of operations between the middleware and mobile client:

A mobile client first sends a request(*send_req_uddi*) to a UDDI registry to discover an existing middleware service. The UDDI registry returns(*send_resp_uddi*) the URI(Uniform Resource Indicator) of the middleware service to the mobile client. The client then sends a request(*send_req*) to the middleware service which includes a URI of its ontology file and a URI of the code to be executed on its behalf. The middleware retrieves the code (*retrieve_code*) and the ontology file (*retrieve_ont*) at the same time and then executes the code (*code_exec*). The

code execution includes requests to the Grid, and thus the rest of the job is done by the Grid. We just simulate it as an immediate transition , as this time would be the same as in the case of a normal Grid user. Upon receiving the results, the middleware scales down the results accord-ing to the device profile in the ontology file (*result_downscaling*) and sends the results to the mobile device (*send_result*). This concludes the communication session. Initially , the place Ready contains a token and at the end of the session the token is in place (*end_session*).

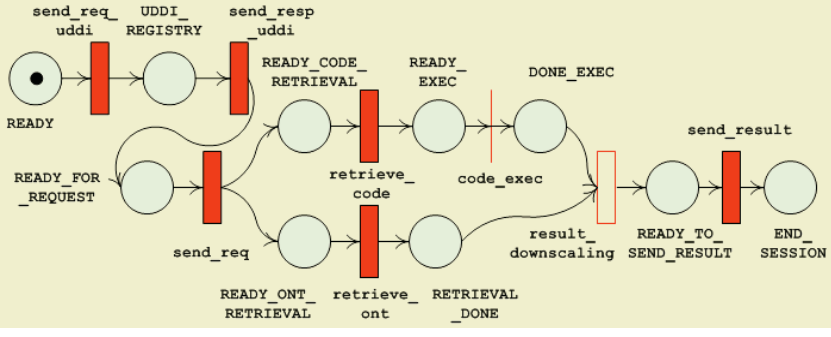


Fig. 2. Petri net model; communication between the client and the middleware

3.1 Parameters Used in the Petri Net Model

In order to make our model simple, we do not consider secure communication as well as disconnected operation. To evaluate the Petri Net model in Figure 2, we used the following numerical parameters which are consistent with the ones used in [1]. We give a description of the parameters as follows:

Size of a request (D_{req})**:** The mobile client sends two types of requests, one to the UDDI (service type request) and one to the middleware service (URIs for the ontology and code files) . The size of these requests can safely be assumed to be small.

Size of the reply (D_{min},D_{max})**:** This depends upon the type of data requested by the mobile client which could be merely a small numerical value (as large as D_{req}) or a little bigger image file. So we have two extremes of data sizes.

Size of the ontology (D_{ont}) and code (D_{code})**:** The ontology file is an XML document and we can safely assume its size to be $\leq$ 10 KB.

Mean processing time for downscaling the results ($1/\lambda_{scale}$)**:** We fix this time as an exponentially distributed random time whose exact value depends upon the specific application.

Throughput of the communication network (Th_{low},Th_{high})**:** We assume two kinds of transmission rates in the network. The wireless network has been

assumed to have lower throughput (Th_{low}), where as the Grid and the middleware service are assumed to be connected with high speed link indicated by (Th_{high}).

We use the above mentioned parameters to describe the distributions associated with the transitions in the Petri Net model, depicted in Table 1.

Table 1. Parameters used in the Petri Net model

Transition Name	Type	Expression
send_req_uddi	Deterministic	(D_{req}/Th_{low})
send_resp_uddi	Deterministic	(D_{min}/Th_{low})
send_req	Deterministic	(D_{req}/Th_{low})
retrieve_ont	Deterministic	(D_{ont}/Th_{high})
retrieve_code	Deterministic	(D_{code}/Th_{high})
code_exec	Immediate	-
results_downscaling	Exponential	(λ_{scale})
send_result	Uniform	$[D_{min},D_{max}]/Th_{low}$

3.2 Numerical Evaluation of the Petri Net

We assigned the following numerical values to these parameters as shown in Table 2 for the evaluation of the Petri Net. These values are consistent with the ones used by [1].

Table 2. Numerical Values used for the Parameters

Parameter	Description	Value
D_{req}	Dimension of client request	1 KB
D_{min}	Minimum amount of Data	1 KB
D_{max}	Maximum amount of Data	30 KB
D_{ont}	Dimension of ontology	10 KB
D_{code}	Dimension of code	40 KB
λ_{scale}	Results scale down rate	4 req/s
Th_{high}	Throughput of wired network	1 Mbps

The firing rate of the *results_downscaling* transition has been fixed to λ_{scale}=4 requests/sec. This factor is not only application dependent but also dependent on the computational power of the computer containing the middleware. However a value of 4 req/sec is a reasonable approximation as used in [1]. We assume a high speed link in the wired network as it constitutes the Grid network and assign a value of Th_{high}=1 Mbps.

Based on these values, we evaluated the Petri Net described in Figure 2 by using the WebSPN [4] [5] tool with which we can associate exponential as well as non-exponential firing rates to the transitions. Figure 3a shows a graph of time

to completion (t)versus the throughput (Th_{low}) of the wireless network ranging from 10Kbps to 1Mbps. The values of the transitions are shown in Table 3. The values for the *send_result* transition has been depicted as [a,b] to show the minimum (a) and maximum (b) value of the uniform distribution.

Table 3. Numerical Values used for the Parameters

Th_{low} K bits/sec	*send_ req_uddi* sec	*send_ resp_uddi* sec	*send_req send_req* sec	*retrieve _ont* sec	*retrieve _code* sec	*results_ downscaling* req/sec	*send_ result* sec
10	0.8	0.8	0.8	0.08	0.32	4	[0.4, 24.0]
20	0.4	0.4	0.4	0.08	0.32	4	[0.4, 12.0]
50	0.16	0.16	0.16	0.08	0.32	4	[0.16, 4.8]
100	0.08	0.08	0.08	0.08	0.32	4	[0.08, 2.4]
200	0.04	0.04	0.04	0.08	0.32	4	[0.04, 1.2]
500	0.016	0.016	0.016	0.08	0.32	4	[0.016, 0.48]
1000	0.008	0.008	0.008	0.08	0.32	4	[0.008, 0.24]

Let's see the affect if we fix the result size to 1KB and the other values the same as in Table 3. We can do that by making *send_result* a deterministic transition with firing rate D_{min}/Th_{low}. After evaluating the Petri Net with this value, we obtain a graph shown in Figure 3b. The graph shows no notable distinction with varying Th_{low}.

We can conclude by studying the two graphs that except for the two obvious parameters, namely the wireless network throughput and the result size, the time to completion is not severely affected by the middleware to client communication and even with low throughput and considerably large result set, the time taken by the middle-ware to mobile device communication is within acceptable limits, only in the order of a few seconds in the worst case.

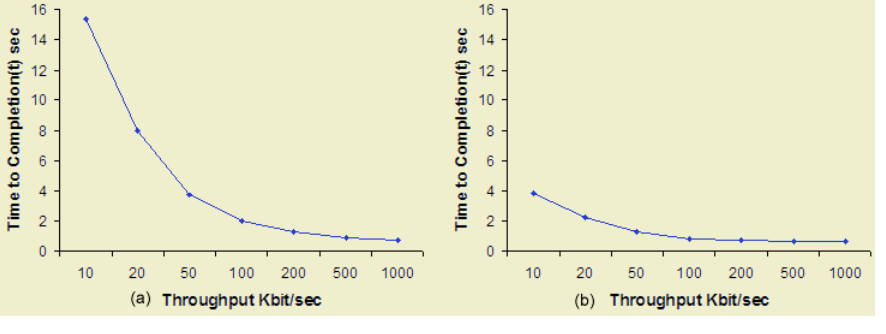


Fig. 3. (a) Time to completion (t) vs Throughput of the wireless network Th_{low}. (b) Time to completion (t) vs Throughput of the wireless network Th_{low} with result size = 1 KB

4 Related Work

Various efforts have been made to solve the problem of mobile-to-Grid middleware. Signal [15] proposes a mobile proxy-based architecture that can execute jobs submitted to mobile devices, so in-effect making a grid of mobile devices. A proxy interacts with the Globus Toolkit's Monitoring and Discovery Service to communicate resource availability in the nodes it represents. The proxy server and mobile device communicate via SOAP and authenticate each other via the generic security service (GSS) API. The proxy server analyzes code and checks for resource allocation through the monitoring and discovery service (MDS). After the proxy server deter-mines resource availability, the adaptation middleware layer component in the server sends the job request to remote locations. Because of this distributed and remote execution, the mobile device consumes very little power and uses bandwidth effectively. Also their efforts are more inclined towards QoS issues such as management of allocated resources, support for QoS guarantees at application, middleware and network layer and support of resource and service discoveries based on QoS properties.

In [16] a mobile agent paradigm is used to develop a middleware to allow mobile users' access to the Grid and it focus's on providing this access transparently and keeping the mobile host connected to the service. Though they have to improve upon the system's security, fault tolerance and QoS, their architecture is sufficiently scalable. GridBlocks [17] builds a Grid application framework with standardized inter-faces facilitating the creation of end user services. They advocate the use of propriety protocol communication protocol and state that SOAP usage on mobile devices maybe 2-3 times slower as compared to a proprietary protocol. For security, they are inclined towards the MIDP specification version 2 which includes security features on Trans-port layer.

5 Conclusion and Future Work

In this paper we identified the potential of enabling mobile devices access to the Grid. We focused on providing solutions related to distributed computing in wireless environments, particularly when mobile devices intend to interact with grid services. An architecture for a middleware layer is presented which facilitates implicit interaction of mobile devices with grid services. This middleware is based on the web services communication paradigm. It handles secure communication between the client and the middleware service, provides software support for offline processing, manages the presentation of results to heterogeneous devices (i.e. considering the device specification) and deals with the delegation of job requests from the client to the Grid. We also demonstrated that the addition of such a middleware causes minimum overhead and the benefits obtained by it outweigh this overhead.

In future we intend to provide multi-protocol support to extend the same facilities to devices that are unable to process SOAP messages. Moreover, we

will continue to focus on handling security, improving support for offline processing and presentation of results depending upon the device. Along with this implementation we intend to continue validating our approach by experimental results.

References

1. Puliafito, A., Riccobene, S., Scarpa, M.: Which paradigm should I use?: An analytical comparison of the client-server, remote evaluation and mobile agents paradigms', IEEE Concurrency and Computation: Practice & Experience, vol. 13, pp. 71-94, 2001.
2. Bobbio, A., Puliafito, A., Telek, M.: A modeling framework to implement preemption policies in non-Markovian SPNs. IEEE Transactions on Software Engineering, vol. 26, pp. 36-54, Jan. 2000.
3. Telek, M., Bobbio, A.: Markov regenerative stochastic Petri nets with age type general transitions. Application and Theory of Petri Nets, 16th International Conference (Lecture Notes in Computer Science 935). Springer-Verlag, pp. 471-489, 1995.
4. Bobbio, A., Puliafito, A., Scarpa, M., Telek, M.: WebSPN: A WEB-accessible Petri Net Tool. International Conference on WEB based Modeling and Simulation, San Diego, California, pp. 137-142, 11-14 January 1998.
5. WebSPN 3.2: http://ing-inf.unime.it/webspn/
6. Hoschek, W.: Web service discovery processing steps. http://www-itg.lbl.gov/ hoschek/publications/icwi2002.pdf
7. UDDI specification: www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
8. SOAP Framework: W3C Simple Object Access Protocol ver 1.1, World Wide Web Consortium recommendation, 8 May 2000; http://www.w3.org/TR/SOAP/
9. GT3 GRAM Architecture: www-unix.globus.org/developer/gram-architecture.html
10. Czajkowski, K., Fitzgerald, S., et al.: Grid Information Services for Distributed Resource Sharing. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
11. Welch, V., Siebenlist, F., et al.: Security for Grid services. HPDC, 2003.
12. Welch, V., Foster, I., et al.: X.509 Proxy Certificates for dynamic delegation. Proceedings of the 3rd Annual PKI R&D Workshop, 2004.
13. Gupta, V., Gupta, S. et al.: Performance Analysis of Elliptic Curve Cryptogra-phy for SSL. Proceedings of ACM Workshop on Wireless Security - WiSe 2002 pages 87-94, Atlanta, GA, USA, September 2002, ACM Press.
14. Giovanni, D., Maryann, H., et al.: Security in a Web Services World; A Proposed Architecture and Roadmap, 2002, IBM and Microsoft Corp. A joint security whitepaper from IBM Corporation and Microsoft Corp. April 7, 2002, Version 1.0.
15. Hwang, J., Aravamudham, P.: Middleware Services for P2P Computing in Wireless Grid Networks. IEEE Internet Computing vol. 8, no. 4, July/August 2004, pp. 40-46.
16. Bruneo, D., Scarpa, M., et al.: Communication Paradigms for Mobile Grid Users. Proceedings 10th IEEE International Symposium in HPDC, (2001).
17. Gridblocks project (CERN) http://gridblocks.sourceforge.net/docs.htm
18. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations, Int'l J. Supercomputer Applications, vol. 15, no. 3, 2001, pp.200-222.

Role of N1 Technology in the Next Generation Grids Middleware

Krzysztof Zielinski, Marcin Jarzab, and Jacek Kosinski

Institute of Computer Science, University of Science and Technology,
Al. Mickiewicza 30, 30-059 Krakow, Poland
{kz, mj, jgk}@agh.edu.pl
http://www.ics.agh.edu.pl

Abstract. This paper addresses problem of the middleware for Next Generation Grid construction and presents how two important concepts such as virtualization and provisioning could satisfy some major requirements of these systems. A vision of NGG is presented and requirements set on middleware layer are analyzed. Next an enabling technology for NGG is considered . The concept of virtualization and provisioning of grid resources is explained. Finally case study illustrating N1 technology application usage is presented.

1 Introduction

There is no doubt that a grid technology will be a very hot area of research and development performed by IT industry and academia during next five years. It is evident from research programs planned by scientific institutions founding research activity like EU IST and declared by major industry companies such as IBM, SUN, HP, Oracle, BEA, Veritas and many others. A coherency of the IT industry strategy and academia should create synergy effect that would result with speed-up of standardization processes and maturity of the available grid technology.

Now a day is observed a gap between commercially available grid solutions and that built and exploited by research communities. The wide acceptance of Web Services [12] as technology for exposing services over internet is not a panacea on existing problems. Also SOA [11] paradigm is not powerful enough to resolve complex task of grid resources management, allocation and software configuration management. The required solution needs more innovative approach related to middleware construction which would exploit a full potential of new operating systems and advance networking technology.

This paper addresses problem of the middleware for Next Generation Grid (NGG) [1] construction and presents how two important concepts such as virtualization and provisioning could satisfy some major requirements of these systems. These concepts built a foundation of N1 [3] technology recently released by SUN.

The structure of the paper is as follows. In Section 2 a vision of NGG is presented and requirements set on middleware layer are analyzed. Next in enabling technology for NGG is considered in Section 3. Finally in Section 4 is presented a case study illustrating N1 technology application usage and power of the virtualization and provisioning techniques employed by this solution. The paper is ended with conclusions.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 942 – 951, 2005.

2 NNG Vision

The NGG report 2004 [2] has identified new additional requirements that have arisen in the light of one more year of experience of the experts working in the Grids domain. In particular, the shortcomings of existing Grids middleware are much better understood, and despite the development directions concerning OGSA [13] providing some integration with services-oriented architectures, it is becoming clearer that applications in the Grids environment require a greater range of services than can be provided by the combination of currently evolving Grids middleware and existing operating systems.

According to [1] NGG will virtualizes the notion of distribution in computation, storage and communication over unlimited resources. A Grid will be a virtual, pervasive organization with specific computational semantics. It performs a computation, solves a problem, or provides service to one single client or to millions of clients. Grids will pervade into everyday life, sometimes in the form of ambient intelligence. Grid may consist of millions of interconnected nodes. A Grid node is an atomic unit forming an abstraction over resources, entailing what is hidden by the interfaces it provides. Nodes may provide new services, functions, or even new concepts that are unknown to clients. The semantics of such services, functions or concepts are defined by semantic description languages or ontology's. Nodes can organize, on the fly, into a group in order to provide functionality and behavior that none of its individual members has. The new "sociology" for a specific service can be transient, persistent, or a combination of both. Any new organization of nodes can be made available to every client of the Grid system.

The self-organizing capabilities of nodes aim at establishing higher robustness and lower costs for systems management. These capabilities are provided through a small, common set of facilities, such as highly scalable protocols for communication and membership management. A new ecology of computers, data entities and communication links will interconnect, interact, interoperate, interfunction syntactically and semantically in a societal way. The dynamics of a Grid manifested by Autonomic Compputing Computing [10] concepts i.e. self-configuration, self-optimization, self-healing and self-protection will allow more freedom for new potential services.

A key aspect of Grids is the capability to negotiate with agents to provide a collection of services and facilities that satisfies the end-user requirement for a price within a certain time. This requires a common understanding of the services and facilities, which necessitate semantic interoperability and a mechanism of assurance to ensure delivery of service trust, provenance, accountability, auditability, traceability.

The next generation Grid will have some mandatory architectural properties:

- Simplicity to allow for easy life cycle management and smooth evolution,
- Subsidiary of control and management, and scalability of services,
- Resilience, through redundant, self-organizing components to minimize points of failure,
- Transparency to allow many virtual organizations to run over it,
- Straightforward administration and trouble-free configuration management.

Describing the programming vision of NGG the report [2] directly points out that to harness the power of an NGG without being overwhelmed by its complexity, abstraction mechanisms must be provided. Those mechanisms will keep all the intricacies of resource allocation and scheduling, data movement, synchronization, error handling, load balancing, etc. transparent to the user and developer. A Grid will possess abstraction layers, presented at the user interface by an agent controlled by metadata, itself interacting with other agents (and metadata) representing the other entities in other Grids via brokers. Future generations of Grids should be programmed through generic and problem-specific abstractions, supported by an appropriate programming environment. In order to achieve the ambitious goal of making all the technical and structural aspects of a Grid transparent, one needs to study and adapt existing programming models to the Grid context – which may require the definition of new programming models, combining parallel and distributed programming practices in a coherent way.

3 NNG Enabling Technologies

The vision of NGG presented by the EC Expert Group Report [2] is still far from implementation. Fortunately the presented concept are very much inline of IT industry effort undertaken by major software and hardware vendors. The NGG concept in many points is coherent with utility computing model present by SUN [9] and other companies.

Successful utility computing programs involve a fusion of technologies, business models and infrastructure management techniques. Specifically, utility computing accomplishes the goal of greater IT resource flexibility, efficiency and value generation through *intelligently matching IT resources to meet business demand on a pay-for-use basis*. *Intelligently matching* refers to the combination of technologies, business processes or applications, and services that will solve a company's specific IT problems. *Business demand* recognizes that a company's business is dynamic and IT is strategic in enabling its success. The *pay-for-use principle* creates real-time connections between IT costs and real-world value.

Several new technologies have enabled this concept and are making it a reality. The technologies are called provisioning, virtualization, and pooling of resources.

Provisioning means making hardware and software available on an as-needed basis. It is the ability to allocate or re-allocate a broad variety of computing resources – including servers, middleware, applications, storage systems and network interfaces to the applications and systems that need them. Provisioning [8] addresses the tasks needed to deploy and activate a service. Whether it's a new operating system or application upgrade, provisioning moves, adds and changes things in a structured, repeatable process. It is a more-efficient form of change management and capacity planning.

Virtualization is a form of resource management, which manipulates devices, storage address spaces, and data objects. Virtualization makes these resources more useable and more effective by aggregating resources together, subdividing resources, and

providing resource substitutions.The most convincing example of virtualization is an operating system which virtualizes computer resources. It is a first elementary example building the ground for higher layer of virtualization. This idea could be applied to grid resources such as computer networks communication, computational nodes and storages.

The abstraction level provided by standard operating systems is not sufficient for contemporary powerful computational nodes. For example Solaris 10 feature, such as Zones [6] formerly N1 Grid containers splits machine into many different operating instances, having own IP addresses, booting capabilities, resources control and accounting, each retaining full security. Zones facilitate the re-provisioning or dynamic partitioning of individual pooled resources, thus offering a pooled environment, with dynamic partitioning of the servers which with no doubt offer a great value for NGG concept.

Pooling of resources means that services are running on a pool of assets, with applications being shifted from one computer (or part of a computer) to another as demand dictates. This situation raises a host of interoperability and configuration management challenges.

4 N1 Technology Case Studies

N1 Grid [3] is Sun's vision, architecture, products and services for optimizing network computing. With the ultimate goal of "managing N computers as 1," N1 Grid comprises solutions designed and built with a common set of architectural principles underlying everything. N1 Grid spans the entire range of network computing. It starts with virtualization technology that delivers "N1 Grid in a box". Building on this foundation, the N1 Grid System provides all the services for managing heterogeneous environments, eliminating the complexity of these processes. When fully implemented, N1 Grid will automate the tasks of resource configuration and dynamic provisioning of capacity to meet fluctuations in the customer's computing load.

In following sub-sections we will describe our experience with N1 Provisioning Server, N1 Provisioning System used for virtualization and services provisioning in the environment of Sun Fire B1600 Blade System Chasis with 48 Sun Fire B100s Blade Servers and Sun Fire B10n Content Load Balancer and Sun Fire B10s SSL Proxy accelerator. At the time of writing this article the Solaris 10 was still in beta release, so we decided to install Solaris 9 on our Blades and that's why advanced resource management features i.e. Zones are not presented.

4.1 N1 Virtualization Techniques – Case Study

The N1 Provisioning Server [4] software provides a comprehensive infrastructure automation solution. This solution enhances the management, visibility, and control of the Sun Fire Blade System. The software allows the complete design configuration, deployment, and management of multiple secure, independent, logical server farms.

These task can be managed using the Control Center, the main component of N1 Provisioning Server software. Control Center is the user interface that enable user to deploy and manage and control logical server farms in a N1 environment.

Every N1 resources are located at an I-Fabric (Infrastructure Fabric) which combines storage, networking and computing resources into a contiguous infrastructure that user can deploy and manage to meet changing requirements. An I-Fabric integrates otherwise individual, heterogeneous servers, networks, infrastructure devices and storage into a coordinated, automated fabric. This fabric enables management, deployment, and redeployment of logical server farms and is made up of three functional areas:

- Control plane – The control plane consists of the N1 Provisioning Server software and the associated server hardware on which the software is deployed.
- Fabric layer – The fabric layer contains the networking infrastructure and switching fabric. This layer enables the software in the control plane to dynamically create, manage, and change the networking and security boundaries of logical server farms. These limits are being realized through creation of dynamic VLANs.
- Resource layer – The resource layer consists of infrastructure resources such as blade servers, load balancers and SSL accelerators.

An I-Fabric devices are controlled and managed by the N1 Provisioning Server Software. The N1 Provisioning Server software comprises two primary components: the N1 Provisioning Server and the Control Center:

- The N1 Provisioning Server is the core automation component and provides the interface between the Control Center and the physical infrastructure resources.
- The Control Center is the web browser-based graphical user interface (GUI) that enables design, configuration, and management of logical server farms, see example depicted in Fig 1.

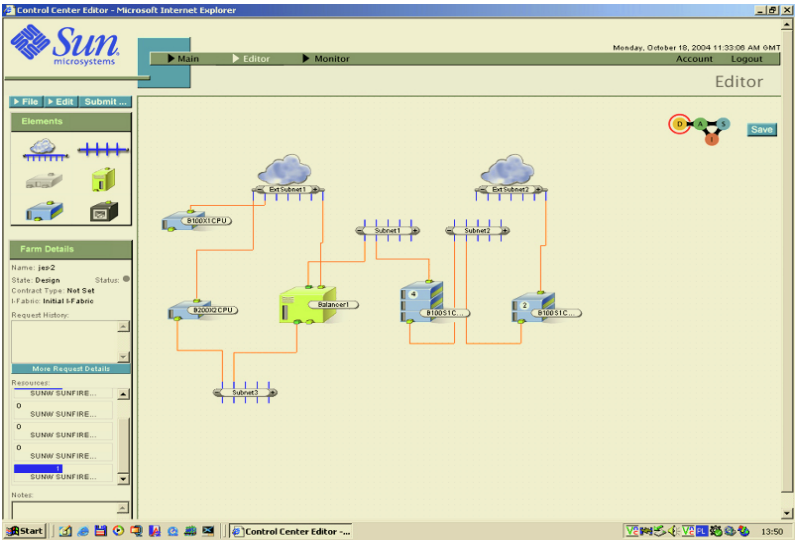


Fig. 1. Control Center Editor – the tool for logical farm creation

The N1 Provisioning Server provides the services required to manage and deploy logical server farms within an I-Fabric, which ensure logical-to-physical mappings

between a logical server farm and the physical resources that are assigned to the server farm.

The Control Center is the user interface to the control plane software that is used to deploy and manage logical server farms. The Control Center enables to do the following tasks:

- Design arbitrary network topology, including subnet configurations,
- Configure and provision servers, load balancers and SSL accelerators,
- Select OS image version for the given Blade node,
- Check for availability of requested resources.

Using Control Center software the logical farm depicted in Fig. 1 has been successfully deployed and activated. It has been used for provisioning case study presented in the following section.

4.2 N1 Provisioning – Case Study

N1 Provisioning System [5] is a software platform that automates the deployment, configuration, and analysis of distributed applications. It enables organizations to manage applications as distinct units, rather than as great set of installation files, thus provides these benefits:

- Centralized control over deployments and configurations,
- Greater coordination among system administrators,
- Increased productivity through automation.

N1 Provisioning System provides an object-model to define a set of Components and Plans for system configuration, service provisioning, and application deployment automation. The N1 provisioning system includes a component library with templates for the most common components in Internet data centers. Operators can capture all the relevant configuration data in a "Gold Server" a.k.a. reference server so that its configuration can be replicated to other servers in a grid.

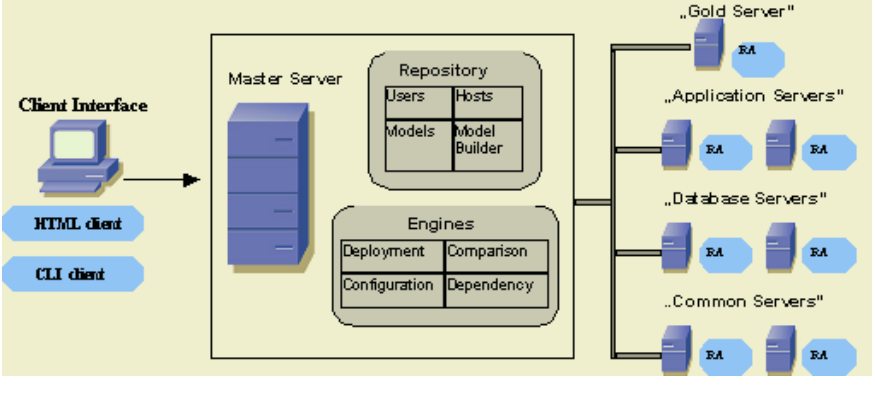


Fig. 2. An overview of N1 Service Provisioning System Architecture

The N1 Provisioning System architecture (Fig. 2) includes a Master Server, which stores component models and operations plans in a secure repository; Remote Agents (RA), are installed on each managed server and perform Execution plans on these servers; and a central console from which IT operators run commands and perform analysis.

In our study we present the deployment process of JIMS (JMX-based Infrastructure Monitoring System) [14] application for Grid Infrastructure monitoring. Monitored resources are computational nodes, storage, network interfaces. JIMS exposes parameters like CPU load, number of processes, memory usage, file system statistics, network interface utilization for the outside world. The main components of JIMS architecture depicted in Fig. 3 are:

- *Monitored Station* (MS) - station (WN - Worker Node) which is being monitored by JIMS. There are gathered information in domains like units of time spent in *user* mode, *system* mode and *idle* mode, memory utilization; buffers, cache sizes.
- *Monitoring Agent* (MA) – process on *Monitored Station* responsible for gathering information and exposing it through JMX/RMI based connectors to *Management Station.*
- *Management Station* – station which performs management role in monitoring the whole Grid infrastructure. Responsible for starting and stopping JIMS services and monitoring agents on *Management Stations* through JMX/RMI interfaces.
- *SOAP Gateway* – one of services of *Management Station.* Acts as proxy between station and client applications, translating requests from SOAP to JMX/RMI.
- *Monitoring Console* – station which is used by system administrator to obtain information about monitored resources. Current version of JIMS supports Java Swing GUI, HTML and CLI (Command Client Interface) clients.

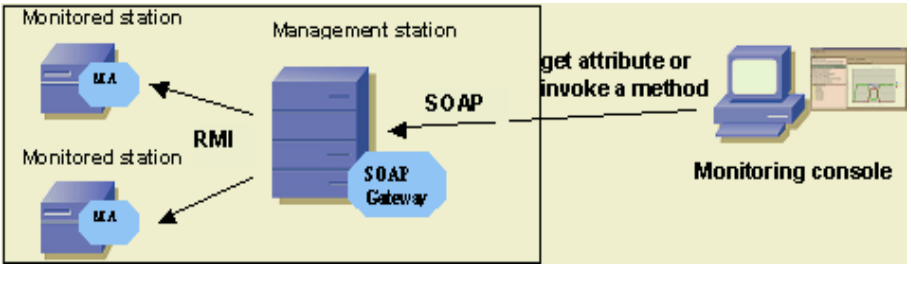


Fig. 3. An overview of JIMS architecture

The N1 Service Provisioning System automates the installation and configuration process of JIMS application. This can be performed in the following activities assuming that, the operating system is Solaris 9.

Activity 1. JIMS must be installed on a *Gold Server* manually. The installation procedure involves several steps performed as a *root* user:

- creation of *cgjims* system group and *cgjims* system user assigned to *cgjims* group,
- creation of directory *$installPath* in which JIMS is to be installed,
- extraction of JIMS bundle into the *$installPath* installation directory,
- changing owner of the *$installPath/var/log/jims* directory to the already created *cgjims* user and group.

By default manual installation process installs all JIMS components including *Monitoring Agent*, *SOAP Gateway* and *Monitoring Console.* In this example it's assumed that *Gold Server* acts as a *Management Station* with installed *Monitoring Agent*.

Table 1. N1 Provisioning System Execution plan used for installation of JIMS

```
1. <component xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
2.         platform='Solaris 9' name='Jims-INSTALL' version='4.1'
3.           xsi:schemaLocation='http://www.sun.com/schema/SPS compo-
nent.xsd'
4.           xmlns='http://www.sun.com/schema/SPS'>
5.         <extends>
6.                   <type name='container'></type>
7.         </extends>
8.         <varList>
9.                   <var name='installPath' default='/opt'></var>
10.        </varList>
11.        <componentRefList>
12.             <componentRef name='ref1'>
13.               <componentn ame='Jims-Cg' path='/' version='1.0'></component>
14.             </componentRef>
15.        </componentRefList>
16.        <controlList>
17.             <control name='run_post_install'>
18.               <execNative userToRunAs='root'>
19.                   <outputFile name='install_jims.out'></outputFile>
20.                   <errorFile name='install_jims_error.out'></errorFile>
21.                   <inputText><![CDATA[chown -R cgjims:cgjims
22.                             :[installPath]/cg/var/log/jims;]]></inputText>
23.                   <exec cmd='sh'></exec>
24.               </execNative>
25.             </control>
26.        </controlList>
27.   </component>
```

Activity 2. In this activity we must create Components and Execution plans which will describe JIMS installation. Our experimental grid environment consists of five SUN

blades which names starts with b100s1gb prefix and N1 Master Server. Host b100s1gb acts as an Gold Server with already installed and running JIMS. Hosts b100s1gb-1, b100s1gb-2, b100s1gb-3, b100s1gb-4 are destination hosts on which JIMS *Monitoring Agents* are to be installed using N1 Provisioning System. To install JIMS following steps are need to be performed by N1 Provisioning System administrator:

- Components and Execution plans creation: *Jims-Cg directory* component which contains information about *Gold Server* repository with JIMS installation bundle; *Jims-INSTALL container* component which is a set of *Jims-Cg* component and Execution plan applied to directory component. Execution plan is featured by an XML script shown in Table1; line 9 specifies *installPath* variable which specifies the target directory for JIMS installation, lines 16-26 depict post installation step named *run_post_install,* which aim is to only changing the owner of *$installPath/var/log/jims* directory.
- Invoking Execution plan (Table 1) on *Jims-INSTALL* component; includes coping of *Jims-INSTALL* to destination hosts to directory specified by *installPath* variable and invoking *run_post_install* procedure described in the Execution plan.

After these activities whole JIMS infrastructure is installed and ready for obtaining monitoring resources in our experimental environment SUN B1600 Blade Chasis with five SUN Fire Blade servers.

5 Summary

The presented analysis of N1 technology fundamental assumptions and NGG reports proposed by group of experts shows the convergence the main concepts and ideas. There is evident that NGG could not be developed only by exploiting the SOA paradigm and relay only on the Web Services. The grid technology requires more fundamental system services built into lower layer middleware and operating systems. The virtualization and provisioning concept illustrated by N1 SUN Microsystems technology illustrates the constructive step in this direction.

References

1. Next Generation Grid(s) European Grid Research 2005 – 2010 Expert Group Report Monday, 16th June 2003
2. Next Generation Grids 2 Requirements and Options for European Grids Research 2005-2010 and Beyond Expert Group Report July 2004
3. N1 Grid System, http://wwws.sun.com/software/n1gridsystem/
4. N1 Provisioning Server Blades Edition, http://wwws.sun.com/software/products/provisioning_server/index.html
5. N1 Provisioning System, http://wwws.sun.com/software/products/service_provisioning/index.html
6. Solaris 10 zones, http://wwws.sun.com/software/solaris/10/inside.jsp

7. SUN Fire B1600 Blade Platform, http://www.sun.com/servers/entry/blade/b1600/
8. Bill Gassman, Provisioning IT Services, Gartner Group, October 2002
9. Utility Computing , Business White Paper SUN Microsystems, March 2004
10. Autonomic Computing IBM Perspective on the State of Information Technology
11. Service Oriented Architecture (SOA) description available at O'Reilly WebServices, http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html
12. WebServices community web page, http://www.webservices.org
13. Open Grid Service Architecture (OGSA) Specification, http://www.globus.org/ogsa
14. JIMS (JMX-based Infrastructure Monitoring System), http://www.eu-crossgrid.org/Seminars-INP/JIMS_monitoring_system.zip

Optimizing Grid Application Setup Using Operating System Mobility

Jacob Gorm Hansen and Eric Jul

Danish Center for Grid Computing,
University of Copenhagen, Denmark

Abstract. This paper is about optimizing Grid application setup by allowing a user to configure a Grid application on her own PC and thereafter migrating the entire application onto the Grid where the Grid system replicates the application including the operating system instance onto the assigned nodes within the Grid. We use this strategy to develop a new Minimal Configuration Grid model (MCG). Configuration is simplified because the user does it on her own machine. Cluster administration is simplified because only a minimal software base is required: No OS nor any shared libraries need be present; they are simply migrated to each node. We have built a prototype MCG based on Intel x86 and the Xen Virtual Machine Monitor.

1 Introduction

Current Grid systems rely on static allocation of jobs to clusters. A Grid user submits a job by proving an application program and a data set which typically are subsequently run on some cluster in the Grid. With such a scheme, there are numerous configuration problems, e.g., the cluster machines must run one of a specific set of operating systems and must have certain libraries present to run the user's specific application. On the cluster side, the local cluster administrator must configure and maintain the OS and the numerous libraries. Besides these configuration problems, current Grid system most often assign jobs to clusters non-preemptively, so if, e.g., scheduling a job requiring 64 nodes on a 256 node cluster, it may be necessary to let some nodes idle until there are 64 nodes available.

In this paper, we address the problems of configuration and non-preemption in Grid systems by introducing a much more dynamic mechanism, that of OS migration, to allow more optimal Grid application setup and dynamic reconfiguration. Our intent is to make job configuration easier for the Grid application writer by allowing the writer to freely configure not only the Grid application itself, but also the writer's choice of operating system. We also intend to reduce the software base necessary on each cluster node thus reducing the maintenance task of the cluster administrator and, at the same time, increasing security by reducing the trusted software base necessary.

Our solution uses virtual machines thus allowing the concurrent execution of several independent Grid application jobs while enhancing security by providing

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 952–960, 2005.

isolation between jobs and the possibility of running multiple independent applications and even independent operating systems on a single machine. We use our implementation of a self-migrating version of Linux, running on top of the Xen Virtual Machine Monitor [1].

We have built a prototype, called Minimal Configuration Grid (MCG), to demonstrate the viability of our ideas. The model includes a low-level token mechanism for secure distribution of usage rights.

1.1 Background

Modern Grid facilities comprise a set of well-provisioned and well-connected machines that are offered to Grid users. In many cases, an entire cluster is dedicated to a single Grid application. There have been recent proposals to use Virtual Machine technology to enable the sharing of Grid resources [2], providing each user with the illusion of having her the machines to herself.

While the motivation for process migration has been transparent balancing of load across cluster-nodes, the techniques necessary for implementing OS migration introduce new possibilities such as network forking where a running OS replicates itself onto a new node on the Grid. Such network forking is greatly eased by the presence of OS migration facilities and is possible with some attention, e.g., to consistency issues.

By migrating entire OS instances we can, in effect, introduce preemption into Grid scheduling systems that previously had none.

1.2 Virtual Machines and Migration

A Virtual Machine Monitor (VMM) is a thin layer interposed between one (or more) OSes such that each OS has its own (virtual) machine which appears to the OS as the actual machine.

Process migration, a hot topic in systems research during the 1980s [3, 4, 5, 6, 7], has seen very little use for real-world applications. [8] surveys the possible reasons for this, e.g., the problem of *residual dependencies* which means that a migrated process still retains dependencies on the machine from where it migrated. In contrast, OS migration does not suffer from the residual dependency problem, because all state is encapsulated within the migrating OS image.

Our migration facility is built on top of the Xen VMM and works as follows: Each virtual machine runs a version of Linux that has migration facilities added and that can migrate *itself* to another machine, we call this facility self-propelled migration [9].

1.3 Minimal Configuration Grid

Our approach is to use the concept of Operating System Migration in the context of Grid systems to build our new model of Grid computing, called *Minimum Configuration Grid*.

The basic idea is for a user to configure the user's own favorite version of an operating system and make the user's Grid application run on the OS. The user submits the job directly to a Grid-node as packaged OS image, or migrates

entire running operating system instance including the user's application to the node.

Furthermore, the Grid can use the migration facility to replicate the OS instance so that the job continues to execute on a number of nodes.

Our results show that, for realistic usage scenarios, it is entirely possible to migrate a production-class operating system between machines on a LAN, with minimal service disruption, typically incurring interruption times down into the $50 - 100ms$ range.

1.4 Contributions

The contributions of this paper are:

- A model, Minimum Configuration Grid, that provides an easy and effective way of handling configuration problems both for Grid users and Grid Cluster administrators.
- A novel Linux/Xen-based implementation of self-propelled replication.
- A low-level token mechanism that allows secure distribution of usage rights.
- Experimental verification that our proposed replication techniques can be effectively and efficiently applied to Grid Computing.

2 Background

2.1 Preemption

With Grid-computing growing in popularity, the limitations of the current generation of job-control software are becoming apparent. The lack of preemptability (e.g., the ability to seamlessly move a running process to either disk or another machine without having to restart it) quickly leads to suboptimal load configurations, because it is impossible to correctly guess in advance the behavior of submitted jobs.

2.2 Configuration

Vast amounts of time is spent on the Grid Community trying to agree on standard configurations for computing nodes, so that jobs may be submitted across clusters at different institutions. Because such standards have to describe both hardware as well as complex software configurations (type of OS, kernel modules for access to network file systems, installed shared libraries, user ids and so forth) and interfaces, deviations are likely. Even if a single software configuration were to be mandated across all institutions, making sure all nodes everywhere are always up to date with the current version of the standard seems like an impossibility.

With OS migration, the need for a common standard is not removed, but because an operating system for a Grid node expects little more than access to a network interface, access to a number of raw disk blocks and memory pages, this

interface is greatly simplified. All that needs to be specified is a protocol for bootstrapping and migrating operating systems, with the rest of the choices being left up to the user. The operating system is then viewed as simply a component of the user's application, in line with a shared library. Because the operating system, in contrast to the normal user process, is entirely self-contained, and because all access to external resources is already abstracted by standardised and fault-tolerant protocols, such as TCP/IP, NFS, or iSCSI, the residual dependency problem of traditional process migration systems is likewise solved.

2.3 Virtual Machine Monitors

The Xen [1] Virtual Machine Monitor is a x86-based VMM that allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion without sacrificing either performance or functionality. Xen provides an idealized virtual machine abstraction that allows OSes such as Linux, BSD, and Windows XP to be ported to it with minimal effort.

In our Minimal Configuration Grid, Xen is used for enforcing isolation between different user applications sharing the same physical hardware.

3 Migration Architecture

Given a VMM-based system and, for example, a Linux guest OS instance running on top of the Xen hypervisor, and a wish to migrate the Linux guest OS to another machine with a similar VMM, we have a number of choices of how to proceed. Our goal is simple; somehow copy the contents of entire virtual machine containing the Linux instance to an empty virtual machine at the destination host, minimising hiccup-time, and with no overlap of execution when viewed from the outside, and in a fail-safe way that will allow us to restore execution on the originating machine, if there is a network partition or other error preventing the migration.

Because live OS migration demands detailed introspection and control of both address space layout and exception handling of the migrating OS, this functionality can only be placed either inside the VMM, or in the guest OS. We have chosen to use an unmodified version of Xen, so that the entire migration is implemented in the guest-OS which in our case is Linux. A typical OS has all the facilities needed for migration, e.g., a TCP/IP stack, fine-grained control of virtual memory, etc. Therefore, it is possible for a guest operating system to perform its own migration. This also simplifies the VMM because it no longer needs to support a full set of migration facilities, for instance it does not need to implement a TCP/IP stack. This has implications for security, as a simpler functionality and smaller implementation of the VMM will make it easier to verify for correctness.

3.1 Self-propelled Migration

Traditional process migration has focused on transparent migration, that is, migrating processes that are unaware that their are being migrated, in order

to allow any process to be migrated. For OS migration, such transparency is not necessarily an advantage. In practice only a few different operating systems are needed by most popular applications, and it may be realistic to modify each of these to support their migration. We therefore propose the concept of *self-propelled OS mobility* as a viable alternative to transparent OS mobility. Self-propelled OS mobility means that the OS not only participates in its own migration, it actually performs the migration itself. There are a number of benefits to self-propelled migration:

- The operating system has close to perfect knowledge of what constitutes its own state including the state of the processes that it runs.
- Much effort has already been put into making popular operating systems efficient at handling the types of operations that are involved in migration, e.g., memory management handling and transmission of large amounts of data over the network.
- The operating system can make better informed decisions during migration, e.g. by rate-limiting processes with large working sets.
- The opportunities for performing other operations such as checkpointing and forking are much better. The operating system can make the often trivial, but essential changes to, e.g., a forked version of itself.

One disadvantage of self-propelled migration is that each OS must be extended to support it; however, once an OS has been extended with self-migration support, the implementation can be ported across multiple VMMs.

From the use of the pre-copy technique in NomadBIOS [10], we know that it is possible to identify the working set of a running operating system, i.e., the pages that are frequently modified during a pre-copy delta cycle, and that this set is both considerably smaller than the full OS. We can start a migration by using pre-copy, and conclude it by using a small buffer for storing the remaining working set by means of copy-on-write. Our self-migration technique thus becomes:

1. Retract all writable page mappings and copy the full OS state to the target machine. Log page frame numbers of modified pages in the page-fault handler.
2. Retract and copy every page that was modified during the last copy.
3. Repeat above step until working set is sufficiently small.
4. Repeat once more, this time making backup copies of any modified pages upon write-fault, and keep the backups in the snapshot buffer.
5. Conclude by copying the backed up pages in the snapshot buffer to their original positions in the image on the target host.

4 Minimum Configuration Grid

4.1 Cluster Configuration

From an administrator perspective, cluster configuration is usually handled using specially crafted tools, cloning a common operating system image (for instance a

popular Linux distribution) onto the harddrives of all nodes. While the process of adding a new node to the network is often fairly automated, the base installation on each node is heavy-weight, often consisting of hundreds of megabytes of persistent state, and so continued maintenance of this persistent state, including the task of keeping all nodes in sync, is cumbersome.

With our system, nodes run only a minimal operating system, with no persistent state on node harddrives. Only the Xen VMM, augmented with a simple mechanism for receiving cryptographic tokens as payment for use, and a small unprivileged bootstrapping executable, reside on each cluster node. All remaining system software and configuration data, for instance, a Linux kernel with a file system populated with shared libraries, executables and configuration files, is supplied by the end-user.

Compared to the current generation of systems, the cluster administrator is relieved of the burden of having to maintain a set of complex OS installations, including the maintenance of cluster-wide state, such as login and password databases.

4.2 Payment for Use

Our self-migration system would work fine in a system without payment. However, we can readily add a mechanism to provide payment for use. Because one of our overall goals is simplicity, we have designed our resource payment system with a simple analogy in mind, namely that of Laundromats. From a customer viewpoint, Laundromats are simple, stand-alone devices that feed on Laundromat tokens. Even though Laundromat sometime fail to deliver the expected service, most customers are still willing to use them, in spite of a small risk of token loss.

For a token-system to withstand abuse, it must be resistant to the following classes of attack:

Counterfeiting. An attacker should not be allowed to spend home-minted tokens.

Double Spending. It should not be possible to spend a token more than once.

There is a number of ways of preventing counterfeiting of tokens. One way is to sign all tokens with the public key of the entity minting the tokens, and have all nodes trust the signature of that entity. Another, and perhaps simpler, method is to use a one-time password scheme, based on a one-way hash function $H()$, which is what we are suggesting. The initial boot-token purchased from and signed by the token-vending service, contains the outermost value T_n in a hash-chain $H^n(T_0), H^{n-1}(T_0), \ldots, H(T_0)$. The rest of the chain is kept as a secret at the token-vending service, and these secrets are only released if the customer pays for them, e.g. using a credit-card or some other form of real-world currency.

During execution, or when attempting to expand its resource allowance, new token-values $T_n, T_{n-1}, \ldots, T_0$ are fed to the VMM on a node by means of a special system call. Token authenticity is simple to verify, by checking that the newly presented token hashes to the previously accepted one.

Because new tokens are minted for each customer-node combination, and because the use of a one-way hash function imposes an ordering of tokens, tokens can only be spent at the node for which they have been minted, and only once. The customer will not purchase an entire chain of tokens at once, but rather purchase tokens little by little, in response to demand from his application and to verification of its progress.

4.3 Security

The security of the system hinges on the isolation properties of the underlying Virtual Machine Monitor (VMM). The problem with most VMMs is, that while they may securely isolate a number of unprivileged guest VMs, it is common to have a privileged host VM running for management purposes. The host VM will usually be sufficiently powerful to pose a danger to the integrity of the guest VMs, in that it has the ability to destroy them or inspect or modify their address spaces. Apart from the VMM itself, which we need to trust, we do not wish to run any code which has privileges over other VMs if it can be avoided, which is one of the main guidelines for our various design choices.

It is clear that we do need *some* privileged functionality, in order to instantiate new VMs, but the actual network stack which is handling the arrival can be created on demand, i.e. upon receival of the first network packet of a new connection initiation. By augmenting connection setup with a cryptographic token, as described in section 4.2, and then instantiate the receiving TCP stack, we can handle most of the setup of new VMs and the receival of migrations without having to resort to privileged code. Only token verification, which is possible with fixed-length buffers and very basic cryptographic algorithms (e.g. a one-way hash function and secret shared between token vendor and Grid node), need to reside inside the VMM or in a privileged VM.

Once a token has been verified, a new VM is instantiated. This VM contains a very basic UIP [11]-derived TCP/IP stack, which handles receiving and final setup of the incoming OS, which will take over the new VM for its own purpose. With this approach, and under the assumption that the underlying hardware and VMM can be trusted, and with the use of the correct cryptographic protocols, it becomes possible to guarantee the integrity of guest VMs.

5 Evaluation

5.1 Bandwidth Test

We measure the overhead of our migration mechanism by looking at the throughput loss of a bandwidth metering test running inside a migrating operating system. From the resulting graph in figure 1, we see that while performance drops during migration, the migrating OS is still able to sustain almost 80% performance. This test is performed across a pair of server-class 2GHz Xeon machines, connected via Gigabit Ethernet.

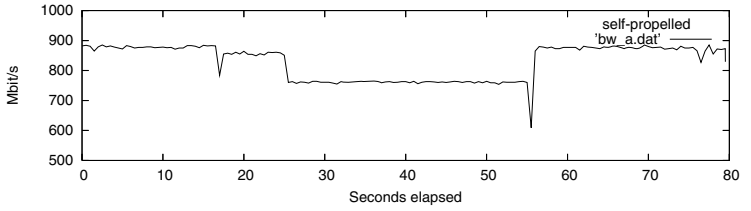


Fig. 1. iPerf bandwidth measure of a Linux VM self-migrating to an equally configured host. The first small drop in performance, at $t = 16s$, is due to migration being initiated, and the final drop, at $t = 56s$, is downtime resulting from execution switching to the new host

5.2 Responsiveness Tests

The purpose of the two remaining tests is to assess service disruption to interactive server processes, when migrating an OS to a different host.

In these tests, we are migrating between an identical pair of workstation-class machines, with 2.4 GHz Pentium 4 CPUs. The machines are connected via switched 100Mbit Ethernet, with Intel EEPro1000 interfaces.

In figure 2 and figure 2, we see that it is possible to migrate both a streaming video server, as well as a quake 2 game server, with minimal disturbance to users. Downtime in this case is about $100ms$. Other tests run across a Gigabit network indicate that with improved network bandwidth it may be reduced below $50ms$.

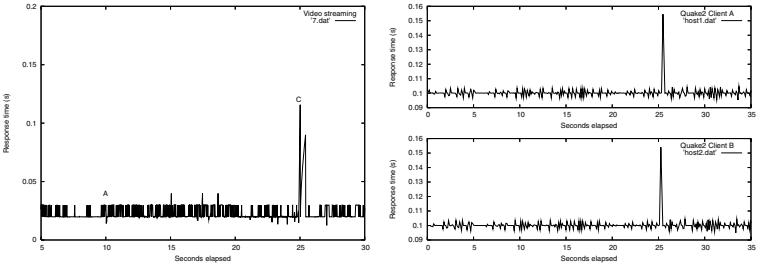


Fig. 2. Left: Downtime experienced when migrating a VLC streaming video server. Final migration occurs at $t = 25s$. Right: Downtime experienced when migrating Quake 2 interactive game server with two clients. Final migration occurs at $t = 25s$

6 Conclusions and Further Work

We have presented a new model for Grid Computing: Minimal Configuration Grid. Our model lets Grid application writers configure their application on their own machines, start up their applications, and when everything is running submit their application to the Grid. The model includes a simple mechanism for resource payment and accounting, it minimizes the trusted computing base of the cluster nodes, reducing configuration complexity and increasing security.

In future work, we plan to expand the prototype onto a 32 node cluster and measure a number of different Grid applications.

References

1. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP19)*, pages 164–177. ACM Press, 2003.
2. Renato J. Figueiredo, Peter A. Dinda, and Jos A. B. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 550. IEEE Computer Society, 2003.
3. Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the emerald system. *ACM Trans. Comput. Syst.*, 6(1):109–133, 1988.
4. Michael L. Powell and Barton P. Miller. Process migration in DEMOS/MP. In *Proceedings of the ninth ACM Symposium on Operating System Principles*, pages 110–119. ACM Press, 1983.
5. Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the V-system. In *Proceedings of the tenth ACM Symposium on Operating System Principles*, pages 2–12. ACM Press, 1985.
6. Fred Douglis and John K. Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. *Software - Practice and Experience*, 21(8):757–785, 1991.
7. A. Barak and O. La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Journal of Future Generation Computer Systems*, 13(4-5):361–372, March 1998.
8. Process migration. *ACM Comput. Surv.*, 32(3):241–299, 2000.
9. Jacob G. Hansen and Eric Jul. Self-migration of operating systems. In *Proceedings of the 11th ACM SIGOPS European Workshop (EW 2004)*, pages 126–130, 2004.
10. Jacob G. Hansen and Asger K. Henriksen. Nomadic operating systems. Master's thesis, Dept. of Computer Science, University of Copenhagen, Denmark, 2002.
11. Adam Dunkels. Full tcp/ip for 8-bit architectures. In *Proceedings of the first international conference on mobile applications, systems and services (MOBISYS 2003)*.

GriddLeS Enhancements and Building Virtual Applications for the GRID with Legacy Components

Jagan Kommineni and David Abramson

School of Computer Science and Software Engineering,
Monash University, Caulfield
Australia, 3800
{jagan, davida}@csse.monash.edu.au

Abstract. The GriddLeS (Grid Enabling Legacy Software) middleware is a novel software layer that allows previously separate legacy applications to be coupled in a software workflow over the grid without any changes to the application source code. We have previously tested GriddLeS on a number of applications, including a small atmospheric sciences workflow in which weather and climate models are coupled. In this paper we describe a number of enhancements to the previous implementation [1][2] that improve its performance significantly. Specifically, the new implementation improves the performance when small blocks are written over high latency networks. The paper also describes a much larger atmospheric sciences workflow than previously reported, that couples multiple global climate models, regional weather models and pollution models in one virtual application.

1 Introduction

To study increasingly complex scientific phenomena of scientific and engineering applications, many new techniques have been developed, ranging from parallel computing to grid computing. Each innovation often requires significant modification to applications to suit new capabilities of the underlying platforms [3][4]. This has slowed down the acceptance and widespread use of new technologies such as MPI [3]. These new techniques often require rewriting an application, which is both costly and error prone. Thus, there is a desire to run applications over the grid without significant modification to the source code. Accordingly, some recent proposals based on standard internet and web infrastructure [5][6][7][8][9] try and leverage existing applications, and minimize the programming overheads in reusing existing code.

Platform neutral protocols for fundamental services like job launching and security are readily available for grid applications [10][11][12][13][14][15]. It is important to abstract various grid capabilities so that users can gain access easily without requiring detailed knowledge about the underlying runtime mechanism, with little or no modifications to applications and further, these abstractions should fullfill application-level needs and expectations [6][16][17][18].

Many scientific applications [19][20] which require data transmission from one application to another in a pipeline. Traditionally these applications run on a single machine sequentially and pass data between them by conventional files. Normally, the

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 961 – 971, 2005.

last application has to wait until the completion of all previous applications because of data dependencies. However, by pipelining these applications and building a virtual application, it is possible to produce results much earlier as each application produces the output progressively. The downstream application can start much earlier than the completion of the previous one. Further enhancements are possible by running different components of the virtual application on various high performance computing systems.

2 The Current Implementation of GriddLeS

The primary use of the GriddLeS environment is to connect applications in a pipeline over the grid and enable access to various resources such as remote files, local files and processes, with a unified interface very similar to the standard POSIX interface. Figure 1 shows some of the different communication patterns that are possible. An application process communicates with files through a web service. The web service acts as a connection point between application processes. That is, inter-process communication occurs by redirecting IO traffic to the web service, which inturn communicates to the file system. The decision about which IO mode is to be used can be delayed until runtime when the files are opened, and need not be coded into the application. It is also possible to copy an entire file from a remote location and this can be opened as a local file. Upon knowing the list of input and output files, the application user enters details such as location of files and mappings in a configuration file. The GriddLeS environment then maps these details at runtime and executes the virtual application. It is also possible to change the mode of transfer and location of files dynamically by closing and reopening files.

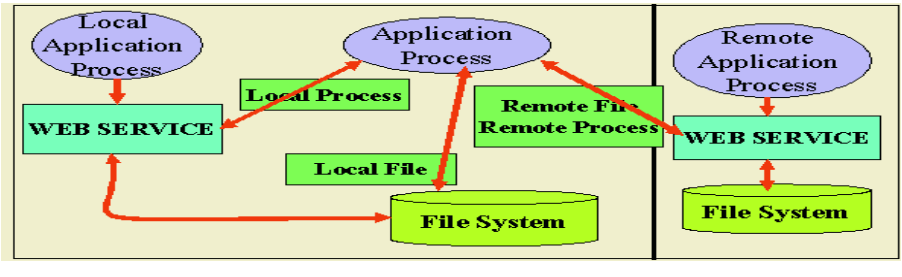


Fig. 1. GriddLeS communication Model

GriddLeS uses FILE IO operations to support inter-process communication and has synchronization inplace when the applications are pipelined. In POSIX, the binding between the physical name of the file in the file system and its file object (inode), occurs when an open call is executed. In the POSIX interface, the metadata of the file is communicated with the stat structure returned from one of the *stat, fstat* and *lstat* system calls. The *stat* interface also gives a hint to the application about the ideal block size for the file operations.

GriddLeS resolves the physical name of a file by using a special naming service, the GriddLeS Name Service (GNS). When the system receives an open call, it assigns

a unique identifier very similar to a file descriptor. In the case of local files it is actually a real file descriptor and in all other cases it is a fictitious one. GriddLeS also obtains the metadata of the file from one of the *stat, fstat* and *lstat* system calls and records this information in a specially created hash table with reference to the file descriptor. In all other cases, GriddLeS mimics metadata with a fictitious local file and modifies its metadata to suit the requirements [1][2]. GriddLes also modifies a preferred block size according to its requirements.

3 GriddLeS Implementation Enhancements

To improve the communication efficiency between the applications, we have made a number of enhancements as follows.

1. Restructure and generalize the grid buffer service.
2. Add a new file status service to improve efficiency.
3. Buffer up small blocks to improve performance when network latency is high.
4. Allow sending and receiving systems to use different block sizes.

Our first implementation of GriddLeS used separate web services for ASCII and Binary data types. A trapped write system call receives binary data independent of the original application data type and passes this data to the web service. On the other hand, a trapped read system call receives binary data from the web service and passes it over to the application module, and the application module handles differences between data types. In the present implementation we use a common web service to act on both ASCII and binary types of data in a unified way. This eliminates overhead in data conversions for ASCII type data. We also eliminated separate web services for file operations by combining the functionality into the GridBuffer web service.

We have added a separate File Status service and it is especially beneficial to reader applications. Before sending the request for data to the GridBuffer web service, the reader application checks the File Status service on the availability of the data. If the data is available then the reader application issues a request for that data. If the data is not available, the reader application check repeatedly for the status until the data is available at the web service by using a special intelligence mechanism. With this mechanism it is possible to increase the time interval between calls to the status web service, when there is no change in status between consequent calls. In the previous implementation all requests from both readers and writers were queued at the web service. This was expensive for the GridBuffer web service to maintain synchronization. This was especially important when the web service was serving many readers and writers at the same time. The Java runtime does not give any guarantee to bring back a specific request from the pool of synchronized requests. Sometimes this may lead to a deadlock situation between reader and writer. In the present implementation, reader requests are not pooled at the web service. This makes the web service respond quickly and avoids deadlock.

In the present implementation we introduced a proxy to help build larger buffers before they are transmitted across the network. This technique is useful when the components of virtual application are distributed to machines in which there is a high

latency between them. For example in our previous case studies, we received poor performance coupling applications between Melbourne (Australia) and Cardiff (UK). The proxy has improved this performance dramatically, as is reported later in the paper. In POSIX, the stat interface allows the operating system to provide an indication about an ideal block size for the file I/O. Most operating systems will give an ideal block size of about 4KB or 8KB. Applications usually send data to files based on this block size. However when the application flushes its data, the data length will be even smaller than the block size. Theoretically once GriddLeS receives data from the application, it can send data to the remote node, however there are some performance implications for the remote file I/O protocol. This results in very large number of connections in a short amount of time. This also consumes considerable of resources on both ends and network devices such as address resolution, on top of the large latency penalties of each activity. We present a case study (case 1 of section 4) by changing block size and measuring time taken to transmit the data. The other significant advantage with this approach is that applications are completely separated from network related problems. In the event of network problems the proxy can be restarted without affecting the application process.

4 Experimentation and Discussions

We used a number of machines to perform experiments as shown in Table 1.

Case 1. Effect of Block size on data transfer time

In this experiment we study the effect of block size on the data transmission time. We consider a proxy script, which reads the data from one file with a block size of 4096 bytes and writes to another file with much bigger block size. The GriddLeS environment is used to configure the experiment. The wall clock times to run the experiments are recorded in Table 2 against block size. In all these cases we transfer the total file (binary data) of size 20MB.

From the results as shown in Table 2 the block size significantly influences the communication time, especially when grid nodes are separated with a high latency

Table 1. Machines List

Name	Address and Details	Country	Name	Address and Details	Country
dione	dione.csse.monash.edu.au Pentium 4, 1500 MHz, Redhat Linux 7.3	AU	brecca	brecca-2.vpac.org 200 Intel Xeon, 2.8 GHz, Redhat Linux 7.3, 124 Node Cluster	AU
mahar	mahar.csse.monash.edu.au Intel Pentium 4, 3 GHz, Debian Linux 2.4.22 50 Node Cluster	AU	freak	freak.ucsd.edu Athlon Processor, 700 MHz,i386, Debian	US
dragon	dragon.vpac.org Intel Pentium 4, 1.70 GHz, Red Hat Linux 8.0	AU	bouscat	bouscat.cs.cf.ac.uk Pentium 3, 1 GHz, Red Hat Linux 7.2	UK

network. On the other hand, if the block size is very big, the writer application needs to wait until the buffer is filled; this may delay the reader application getting access to the data.

Table 2. Effect of block size (in bytes) on data transmission time (in min:sec)

Block size	dragon-dione time	dragon-freak time	dragon-bouscat time
4096	1:15	50:23	1:21:11
10240	1:13	27:07	51:09
20480	0:41	16:52	32:09
40960	0:39	10:11	22:01
61440	0:39	07:04	16:56
102400	0:38	05:29	13:44
204800	0:37	03:54	07:59
307200	0:36	03:13	07:47
409600	0:36	03:00	07:03
512000	0:36	02:49	06:51

The results indicate the increase in block size beyond 204800 may not improve performance significantly, hence we adopted block size of 204800 in the future experiments whenever the proxy is involved.

Case 2. Atmospheric Models

In this case study we consider various atmospheric models and demonstrate the performance of the new system. We also show a much larger atmospheric sciences case study than has been presented before [2]. A global climate model (GCM) [19] is a computer model derived from mathematical equations based upon the laws of physics for representing the atmosphere, oceans, land and sea-ice to simulate the behaviour of the climate system. All Global climate models capture large scale features like deserts and tropics very well, but have difficulty in capturing small scale features like cyclones and thunder storms because they occur on a smaller scale than the grid used. A regional climate model (RCM) [20] with horizontal resolution of 100 km or less, is able to simulate regional weather pattern better than most GCMs. Since topographic features strongly influence regional temperature and rainfall, more detailed features are likely to give a better prediction with RCM. An RCM needs metrological information at its grid nodes from the fine resolution stretched GCM. A fine resolution stretched GCM requires metrological information at its grid nodes from the coarse resolution GCM which inturn gets metrological information from a uniform resolution GCM. The CIT (California Institute of Technology) Model is a photochemical air quality model [21] and has been designed to estimate the formation and transport of photochemical air pollution. The RCM predictions are used by CIT model to drive the process of pollutants transport and depositions, and to provide radiation, temperature and moisture fields for the simulation of chemical transport process.

The column heading “with Files” for tables 4 and 5 indicates the results are obtained when models run sequentially using conventional files that are copied from one

system to another. The words "with Buffers" in a column heading for tables 4 and 5 indicate that the results are obtained by connecting models using GriddLeS middleware. The GriddLes middleware acts as a communication mechanism between the applications. To achieve this the normal IO primitives in conventional languages have been overloaded so that they support interprocess communication as well as file operations. This makes the individual components behave as if they are operating in a conventional file system, but in fact they are sending and receiving data across a distributed grid infrastructure.

Case 2a. Global and Regional Models with proxy

In this case, as shown in Figure 2, we consider a Global Climate Model and a Regional Weather Model and run over grid nodes where the network latency is varied from local to very distant as shown in Table 3. In this experiment all the machines are single CPU Linux machines. We run this experiment for the range of time steps from 960 to 1920 and the results are printed at every 60^{th} time step in all the test cases.

Fig. 2. Global and Regional Model pipeline

Table 3. Latency Measurements

Data size in bytes	Node 1	Node 2	Round trip time (ms)
64	dragon	dione	2.13
64	dragon	freak	196
64	dragon	bouscat	330

We run the Global Model (CCAM), cc2lam (a small conversion program) and the proxy on dragon as independent processes and the Regional (DARLAM) on another machine. The proxy reads the data produced by cc2lam and writes into another file with a much bigger block size (204800).

The results shown in Table 4, indicate that it is possible to hide the entire network latency even though the latency between two grid nodes is significant and we are able to gain performance very similar to low latency network connections. This is possible because the application execution and data transfer occur in parallel. The proxy is especially advantageous in the case of distant systems. The results without the proxy did not give much improvement in performance in comparison with the conventional methods. This is due to more communication overheads because of the large numbers of connections with smaller block sizes.

Table 4. Models run between high latency grid nodes

Model	Machine	960 time steps			1440 time steps		1920 time steps	
		with Files	With Buffers		With Files	With Buffers	With Files	With Buffers
			Proxy	No Proxy				
C-CAM	dragon	1:01:31	1:04:41	1:05:10	1:33:57	1:37:01	2:03:32	2:09:32
cc2lam	dragon	1:01:57	1:05:13	1:13:44	1:34:14	1:40:14	2:04:12	2:10:02
F Copy	dragon	1:03:34	-	-	1:36:39	-	2:07:17	-
Proxy	dragon	-	1:05:47	-	-	1:40:51	-	2:10:39
Darlam	dione	1:30:53	1:10:39	1:15:22	2:17:20	1:42:15	3:01:29	2:21:45
C-CAM	dragon	1:01:31	1:04:28	1:05:21	1:33:57	1:36:17	2:03:32	2:11:18
cc2lam	dragon	1:01:57	1:05:06	1:29:11	1:34:14	1:37:50	2:04:12	2:11:53
F Copy	dragon	1:06:46	-	-	1:41:17	-	2:13:22	-
Proxy	dragon	-	1:05:41	-	-	1:39:20	-	2:14:06
Darlam	freak	1:34:12	1:16:07	1:30:28	2:22:07	1:42:10	3:08:14	2:17:00
C-CAM	dragon	1:01:31	1:04:47	1:05:33	1:33:57	1:36:28	2:03:32	2:10:37
cc2lam	dragon	1:01:57	1:05:16	1:52:21	1:34:14	1:36:33	2:04:12	2:10:44
F Copy	dragon	1:10:22	-	-	1:46:51	-	2:20:47	-
Proxy	dragon	-	1:10:55	-	-	1:41:59	-	2:16:04
Darlam	bouscat	2:14:06	1:23:36	2:10:28	3:22:27	2:16:33	4:28:15	2:39:57

Case 2b. Multi-model Pipeline application running on different systems

In this experiment, we consider all five models running on various systems. The first global model uses a uniform grid spacing of 300km, whereas the other global models vary the resolution of the grid to give more accuracy in the area under consideration (i.e., non-uniform grids of spacing 60km to 800km and 6km to 6000km). The RCM uses a uniform grid of 6km x 6km over the region of 100km x 100km. The regional model produces an output which, in turn is used by the CIT model.

From the examples shown here, it is clear that GriddLeS allows us to create a virtual application using existing atmospheric models with almost no modifications. The composed virtual application consists of a number of atmospheric models, Global climate models with both uniform and non-uniform grid spacing, a regional model (DARLAM) and a pollution model (CIT) and their interfaces, run in a pipeline in a grid environment. Traditionally, these models have been run sequentially on the same computer. Binary data of about 72 MB (for every 480 time steps when results are printed at every 60th time step) is passed between them by using conventional files. These models also use many other local files specific to each model. Most of these models are computationally intensive and it is possible to overlap computation by connecting these models with GriddLeS runtime and transferring data at the same time when the model is performing other parts of the computation. Downstream applications in a pipeline can start much earlier than their counterpart in sequential runs.

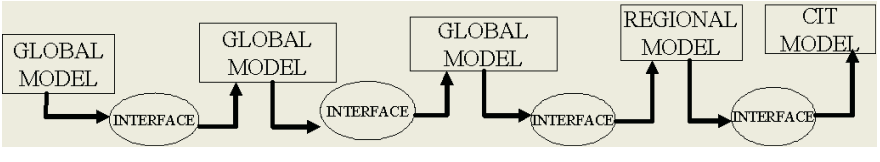


Fig. 3. Virtual Application

The synchronization between models and data transmission from one model another is handled by the GriddLeS runtime automatically, without the user's involvement.

The present virtual application consists of 9 independent processes running on 5 different grid nodes located over 4 different organizations as shown in Figures 3 and Table 5. We used a proxy whenever there is a need to transmit data from one machine to another. The proxy script runs as a stand alone process independent of the rest of the application processes. Currently, all of the nodes in particular virtual application obtain their configuration information from a central node which runs a GriddLeS Name Service (GNS). The GNS tables contain a number of entries – one per file. Each entry holds information about whether the file is found locally, whether it is remote or whether it pertains to a remote pipe. At present, these entries are written manually by a user when they configure a virtual application. However, in the future, we plan to use a workflow generation tool like Kepler [22] or Triana [23] to do this.

We used globus [10] middleware to submit jobs to different grid nodes. All jobs were submitted at the same time. The GriddLeS runtime can handle the rest of the activity. Reader applications will be blocked if there is not sufficient data, however writer applications will continue execution without any interruptions. Eventually this will also be replaced by a grid workflow tool.

In this experiment the various models are distributed to different computing systems as shown in Table 5. In the case of sequential runs, files are copied over to the next system after the completion of the current model execution using the `scp` command. In the case of buffers, all models are distributed and run in parallel at the same time. Operations such as synchronization between the models and data transfer from one node to another are performed at runtime automatically, without the user interaction. The results indicate that the application with GriddLeS runs much faster than its counterpart with conventional files.

Table 5. Models Distributed on to different machines

Model	Model run location	Results location	With Files	With Buffers
Global	dione	dione	03:25	06:13
Interface	dione	dione	03:46	06:29
File Copy (scp)	dione	dragon	04:51	-
Proxy	dione	dragon	-	07:04
Global	dragon	dragon	10:26	11:20
Interface	dragon	dragon	10:44	11:42
File Copy (scp)	dragon	brecca-2	10:55	-
Proxy	dragon	brecca-2	-	12:12
Global (CCAM)	brecca-1	brecca-2	27:21	24:37
Interface (cc2lam)	brecca-1	brecca-2	27:37	24:46
Regional(Darlam)	brecca-2	brecca-2	35:15	27:55
Interface	brecca-2	brecca-2	35.26	27:57
File Copy (scp)	brecca-2	mahar	35:46	-
Proxy	brecca-2	mahar	-	28:22
CIT	mahar	mahar	1:06:10	48:15

5 Conclusions

In this paper we have discussed enhancements to the GriddLeS runtime environment that enables programs to run as components of a grid application (virtual application) without changes to the source code of underlying components, which are themselves applications. These legacy applications were written in Fortran programming language and were designed without any knowledge of the distributed computing environment. We have shown the simplicity of stitching individual components to make a complex virtual application.

One of the most significant achievements from the experiments is that for a special class of problems where execution is more significant, the latency can be hidden completely, even though the applications run over grid nodes which are separated by high latency network. The performance improvements in these cases are very similar to low latency network connections. GriddLeS makes use of existing software like Globus [10], bypass[5], JWSDP [24] from Sun and gsoap[25][26] amongst others. This is viewed as a complement to existing grid middleware to deploy existing applications over to the grid environment without source modification.

Acknowledgements

The authors would like to acknowledge the work of CSIRO Scientists, in particular Drs. Jack Katzfey, Martin Cope and John L. McGregor who provided Atmospheric Sciences case studies. This work is supported by Australian Research Council and Hewlett Packard under and ARC Linkage grant.

References

[1] J. Kommineni, "Grid Applications: A Seamless Approach with Web Services", The APAC Conf. and Exhibition on Advanced Computing, Grid Applications and eResearch Royal Pines Resort Gold Coast, Queensland Australia. 29 September - 2 October, 2003.

[2] D. Abramson, and J. Kommineni, "An Atmospheric Sciences Workflow and its Implementation with Web Services", The International Conference on Computational Sciences, ICCS04, Ktakow Poland, June 6 – 9, 2004.

[3] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and D. Dongarra, MPI: The Complete Reference, Published by the MIT Press, 1995.

[4] J. J. Dongarra, and D. W. Walker, "MPI: A Standard Message Passing Interface", Supercomputer 1996, pp. 56-68.

[5] D. Thain, and M. Livny, "Multiple Bypass: Interposition Agents for Distributed Computing", Journal of Cluster Computing, Volume 4, 2001, pp. 39-47.

[6] D. Abramson, and J. Kommineni, "A Flexible IO Scheme for Grid Workflows", IPDPS-04, Santa Fe, New Mexico, April 2004.

[7] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development", International Journal of High Performance Computing Applications, Winter 2001 (Volume 15, Number 4), 2001, pp. 327-344.

[8] G. Allen, D. Angulo, T. Goodale, T. Kielmann, A. Merzky, J. N., J. Pukacki, M. Russell, T. Radke, (Ed Seidel, J. Shalf, I. Taylor), "GridLab: Enabling Applications on the Grid: A Progress Report", In 3rd International Workshop on Grid Computing in conjunction with Supercomputing 2002, LNCS Vol. 2536, 2002, pp. 39-45.

[9] Y. Huang, I. J. Taylor, D. W. Walker, and R. Davies, "Wrapping Legacy Codes for Grid-Based Applications", in Proceedings of the 17th International Parallel and Distributed Processing Symposium (Workshop on Java for HPC), held 22-26 April 2003 in Nice, France, ISBN 0-7695-1926-1.

[10] I. Foster, and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Supercomputer Applications, 11(2), 1997, pp. 115-128.

[11] I. Foster, and C. Kesselman, (editors), The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition, Morgan Kaufmann, 2004. ISBN: 1-55860-933-4.

[12] Condor DAGman, http://www.cs.wisc.edu/condor/

[13] S. Chapin, J. Karpovich, and A. Grimshaw, "The Legion Resource Management System", Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 1999. (http://legion.virginia.edu/).

[14] R. Buyya, D. Abramson, and J. Giddy, "An Economy Driven Resource Management Architecture for Global Computational Power Grids", International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, Nevada, USA, June 26 – 29, 2000.

[15] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003. ISBN: 0-470-85319-0.

[16] D. Thain, and M. Livny, "Bypass: A tool for building split execution systems", In the Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, Pittsburg, Pennsylvania, August 1-4, 2000, pp. 79-85.

[17] D. Thain, and M. Livny, "Parrot: Transparent User-Level Middleware for Data-Intensive Computing, Workshop on Adaptive Grid Middleware", New Orleans, Louisiana, September 2003.

[18] D. Cheriton, "UIO: A Uniform I/O system interface for distributed systems", ACM Transactions on Computer Systems 5(1), 1987, pp. 12-46.

[19] J. L McGregor, and J. J. Katzfey, "NWP experiments with a variable-resolution conformal-cubic primitive equations model", In: Research activities in atmospheric and oceanic modeling, A. Staniforth (editor). (CAS/JSC Working Group on Numerical Experimentation Report; 27; WMO/TD - no. 865) [Geneva], 1998.

[20] J. J. Katzfey, and J. L. McGregor, "Verification and evaluation of storm tracks in regional climate simulations over Australia", In 10th Sym. on Global Change Studies: preprints, Dallas, Texas. Boston: American Meteorological Society, 1999, pp. 384-386.

[21] K. J. Tory, M. E. Cope, G. D. Hess, S. H. Lee, and N. Wong, "The use of long-range transport simulations to verify the Australian Air Quality Forecasting System. In: Modelling and predicting extreme events": extended abstracts of presentations at the fourteenth annual BMRC Modelling Workshop, BMRC, Melbourne, A. J. Hollis, and P. J. Meighen (editors) (BMRC Research Report, 90) Melbourne: Bureau of Meteorology Research Centre., 2002, pp. 19-23.

[22] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock, "Kepler: Towards a Grid-Enabled System for Scientific Workflows", In the Workflow in Grid Systems Workshop in GGF10 - The Tenth Global Grid Forum, Berlin, March 2004.

[23] S. Majithia, I. Taylor, M. Shields, and I. Wang, "Triana as a Graphical Web Services Composition Toolkit", in Proceeding of the UK e-Science Programme All Hands Meeting 2003, held 2-4 September 2003 in Nottingham, UK, edited by S.J.Cox.
[24] Jwsdp-1.3, "Java Web Service Toolkit" http:// java.sun.com/webservices, 2003.
[25] R. van Engelen, "The gSOAP toolkit",http://www.cs.fsu.edu/~engelen/soap.html, 2003.
[26] R. van Engelen, K. Gallivan, G. Gupta, and G. Cybenko, "XML-RPC agents for distributed scientific computing", In IMACS'2000 Conference, Lausanne, 2000.

Application Oriented Brokering in Medical Imaging: Algorithms and Software Architecture*

Mario Rosario Guarracino[1], Giuliano Laccetti[2], and Almerico Murli[1,2]

[1] Institute for High Performance Computing and Networking - Naples branch, National Research Council, Italy
[2] Department of Mathematics and Applications, University of Naples Federico II, Naples, Italy

Abstract. This paper describes algorithms and software architecture of a resource broker designed in the context of MedIGrid, a medical imaging application for the management, visualization and reconstruction on grids of medical images produced by PET/SPECT medical instruments. The broker allows the discovery and selection of suitable clusters of workstations for the execution of parallel image reconstruction algorithms. The proposed algorithms and software architecture are general with respect to possible application domains and are potentially useful in different grid environments.

1 Introduction

Grids are distributed platforms in which heterogeneous machines can be accessed through a single interface. The underlying computational model is a distributed environment in which hardware, software and configuration resources, belonging to different organizations, can be shared among authorized users. Resources dynamically publish ads about their current status in proper registries and periodically update such information. Resources are shared between users; each institution manages and administers its own resources autonomously. A user can use available resources within the policy that are set by owning institution.

In that context the term *broker* refers to a tool with a set of functionalities that enable *discovery* and *selection* of resources in a distributed computing environment for one particular application, with specific needs. Discovery and selection are accomplished by a match between application and user's needs, and information obtained in the registries. Let us suppose a user wants to run his SPMD application on a set of computing resources, using an input data set and storing the results in a database. The user requirement is to have the results back within a specified period of time; an application requirement could be, for example, a given operating system. In the outlined computational model, the broker needs to discover in the registers resources capable

* This work has been partially supported by Italian Ministry of Education, University and Researh (MIUR), within the activities of the WP9 workpackage "*Grid Enabled Scientific Libraries*" coordinated by A. Murli, part of the MIUR FIRB RBNE01KNFP *Grid.it* Project *"Enabling Platforms for High-Performance Computational Grids Oriented to Scalable Virtual Organizations"*.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 972 – 981, 2005.

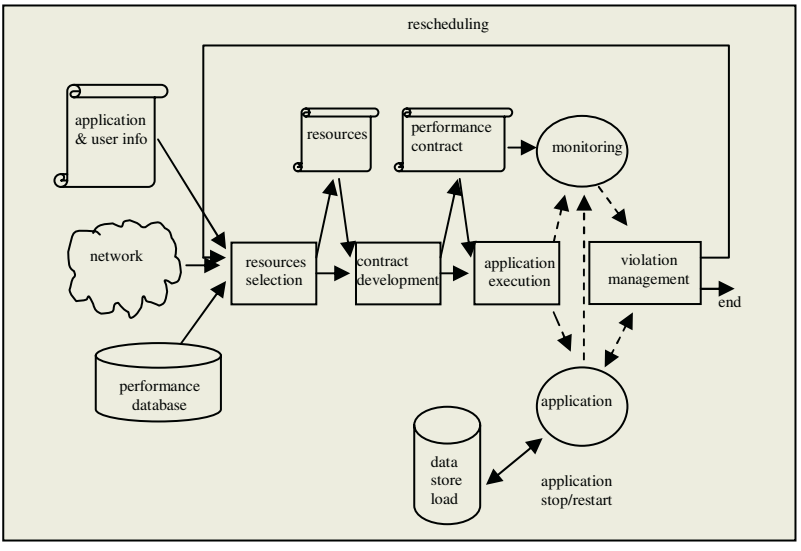


Fig. 1. Application manager workflow

of solving the problem and to select one of them to satisfy user requests. This example shows needed information regards hardware requirements (a minimum amount of main memory, connection network bandwidth, ...), software requirements (a particular operating system, pre-installation of a particular software, ...) and configuration requirements (existence of a distributed file system among the computing elements, ...). Once the resources have been selected, they need to be monitored to check if they are delivering the required computational power, in order to ensure user requirement are met. An actual Performance Contract will be defined as described in [5]. If that is not the case, a rescheduling phase can take place, in which the broker is again in charge to find new resources that can in principle solve the problem. In case of application rescheduling, part of the previous elaboration can be probably reused; since initial conditions are different, broker output will be, in general, different. This situation is depicted in **Figure 1**.

Our interest in grid computing brokering is motivated by a previous work on MedIGrid [2], a distributed application for the management, processing and visualization of biomedical images that integrates a set of software and hardware components, or, more specifically, a set of grid collaborative applications useful to nuclear doctors. It results from the interaction of scientists devoted to the design and deployment of new tomographic reconstruction techniques, researchers in the field of distributed and parallel architectures and physicians involved in neural medicine. The main drawback of reconstruction methods is that they are so computationally intensive that cannot be employed on conventional machines. MedIGrid uses MedITomo [3], a parallel software for reconstruction of SPECT images based on a conjugate gradient computational kernel. The medical doctor selects, within the volume of acquired data, the slices of interest and initializes parameters related to the accuracy of the reconstruction process. That means the computational cost of a single reconstruction only depends on the number slices to be reconstructed and iterations to be performed, which makes it possible to model the expected execution time over different computing resources.

On the other hand, high performance computing technology can be employed in SPECT devices only to an extent which is determined by the overall cost of the system and its localization in *ad hoc* places. Delocalization of acquisition instruments from processing power and storage facilities seems a viable way to overcome such difficulties. Indeed, end-users of such applications are not "experts" of distributed computing management; it becomes mandatory to hide as much as possible difficulties related to use of high performance geographically distributed platforms and to manage, catalogue and process this huge amount of data. Moreover, using computational grid technology can solve problems related to authentication, dynamic allocation and other aspects connected to remote resources access.

Finally, the application of the tomographic concept to Nuclear Medicine data led to powerful new techniques such as Single Photon Emission Computed Tomography (SPECT) and Positron Emission Tomography (PET). The introduction of these techniques represents the core of Nuclear Medicine methodologies. In all cases, the aim is to obtain the spatial distribution of the physical quantity to be imaged from the measured data (the so-called raw data) (see, again, [2]).

Since that experience, situation has much evolved and, although it realized an economy of thought, software tools now available in Globus middleware (http://www.globus.org) open new scenarios for the medical community to collaborate and share resources, information and knowledge. The application can now take advantage of information provided by the user, previous executions and network monitors to discovery and select computing resources that can reconstruct raw data acquired by PET/SPECT medical instruments within a given period of time. Therefore, it has become necessary to introduce algorithms and software architecture for brokering resources for the application and it is more and more urgent as size, users and number of available grid testbeds increase, since the application can efficiently perform only if available resources can be selected with respect to users needs.

There are basically two approaches for the brokering problem:

1. *Application oriented approach*: the broker is embedded in the application and it tries to minimize the execution time of a single application execution on a set of potentially shared resources. The information is application dependant and the use of the broker in another application is not straightforward.
2. *Runtime scheduling:* the scheduler tries to maximize the overall system performance, considering the performance of the single application. The scheduler takes care of the different applications running at the same time on the system, trying to optimize resource utilization.

The first approach is used, for example, in the GrADS project [8], in which the scheduler uses a performance model of the application to decide the mapping of the single application components to the discovered resources. An example of the second approach is Prophet [15], a run time scheduling effort which targets heterogeneous systems and includes parallel applications with inter processor communications. In present scenario the notion comes of meta-scheduler [13], which allows to request resources from more than one machine for a single job [12]. It may perform load balancing of workloads across multiple systems, in which case each system would then have its own local scheduler to determine how its job queue is processed. It requires advance reservation capability of local schedulers. In those strategies either informa-

tion about the underling environment is passed to the application or application information is passed to the runtime support. In both cases solution is application dependent and it is difficult to use in different setting.

In the following we describe MedIBroker, a grid broker targeted to clusters. It is a tool to search a grid testbed and find a cluster that can be used to run an SPMD image reconstruction application within a given period of time. It tries to couple the previously described approaches merging application information with one related to the computing environment. The broker is application oriented and it uses information about the computing environment that is provided by registers and a history database of application runs. The strategy has the advantages of application oriented brokering, that is relieving the runtime support from a task that can be resource and time consuming; moreover it provides hints on resource management that are application specific but do not require any knowledge about the application itself, which makes it portable to different applications.

2 State of the Art

The discovery and selection algorithms have already been investigated in metacomputing environments. A review can be found for example in [7]. Those topics assume a key role even in a grid computing environment, and they have to be investigated as well. In the following we try to summarize a certain number of grid computing projects in which a broker has been developed. It does non mean to be exhaustive in any way, but rather to give an idea of the efforts devoted to the topic in major projects around the world, providing, in **Table 1**, URLs for other interesting projects. See also [8] for further details.

Table 1. Brokers in grid software projects

Cactus	PSE	http://www.cactuscode.org/
DISCOVER	Telecollaboration	http://tasslweb.rutgers.edu/discover/main.php
Entropia	Application support	http://www.entropia.com
EuroGrid	Integration	http://www.eurogrid.org/
GRB	Framework	http://sara.unile.it/grb/
GridLab	Toolkit	http://www.gridlab.org/
Griphyn	petabyte data grid	http://www.griphyn.org/index.php
GridPort	Portal development tool	http://gridport.sourceforge.net/
Narada	Broker	http://www.naradabrokering.org/
Nimrod	Parameter swapping	http://www.csse.monash.edu.au/~davida/nimrod/
Unicore	IDE	http://www.unicore.org/forum.htm
GridSim	Scheduling simulator	http://www.buyya.com/gridsim/

Apples (Application Level Scheduler) [6] is a software package that allows the scheduling and deployment of medium or large scale parameter sweep applications over computational grid resources. It interfaces with many grid middleware for job submission, data movements, and resource monitoring. Its broker is targeted to those problems and selects resources from available ones.

Condor-G [9] manages both a job queues resources from one or more sites where those jobs can execute. It communicates with these resources and transfers files to and from these resources using Globus mechanisms. Condor-G uses the GRAM protocol for job submission, and it runs a local GASS server for file transfers. Condor-G is not a simple replacement for the Globus toolkit's *globusrun* command in that it allows the submission of many jobs at once, the monitoring of those jobs with a convenient interface, the notification when jobs complete or fail, and the management of Globus credentials which may expire while a job is running.

GrADS [14] takes its steps from Apples project and it implements a software architecture to support monitoring and performance adaptability on grids. Monitoring is intended as the capability to sense application performances during its execution, while adaptability is the ability to dynamically change computing resources during the elaboration, in order to maintaining an a priori agreed level of performance.

DataGrid [10] is an EU project for a software infrastructure for next generation high energy physics experiments. The workload management section of DataGrid has the goal of defining and implementing an architecture for distributed scheduling and resource management in a grid environment. This infrastructure tries to cope with an unpredictable, chaotic workload generated by relatively large numbers of independent users in contrast to the supercomputing environments that have been the target of most previous meta-computing projects.

Netsolve, and its evolution **GridSolve**, aim is to implement the middleware to bridge simple application program interfaces, which are usually provided by problem solving environments, and the set of services provided by grid middleware software tools. Netsolve can take advantage of Condor tools to discover and manage available resources [1].

Ninf-G is an implementation of GridRPC [11] programming model on top of Globus. It provides the ability to run applications on grid, with a stub mechanism to remotely run executables, providing application programming interfaces and registration mechanisms that enable to register software components within the Globus Metadata Directory Service.

3 MediBroker

The broker has mainly two functions: discovering and selecting resources in a testbed that can better fulfill application and user requirements. In the first step, a search in the information system results in the discovery of a set of available machines. This phase provides the characteristics of resources found, some of which, such as CPU speed and operating system type, can be considered static, and others that can vary during time, such as CPU load and memory utilization. A static property is one for which the probability it changes in the period between its acquisition and use is low.

Discovery procedure takes as input static characteristics describing the needed resources and performs a search in the register as outlined in **Algorithm** 1.

```
Procedure discovery (OS type, MEM-size-test, register)
list=empty

for machine in register

        MEM_avail = Mds-Memory-Ram-availMB
        if (MEM_avail   >= MEM-size-test  &&
 OS type == Mds-Os-type)
        Add machine to list
      end if
 end for
return list
```

Algorithm 1. Discovery

The input data are operating system type on which we want the application to run, the minimum amount of memory, and the Globus MDS server to search. The procedure returns a list of available clusters that match the requirements. This strategy works for a small number of resources since its complexity grows linearly with the number of resources. An alternative solution could relay in a user list of resources, dynamically updated by the broker, which have been successfully used to solve the problem. The broker is targeted to homogeneous clusters with all processors of the same family, although each cluster could have different processor speeds. If a cluster has groups of CPUs of different families, each of those has to be considered as a different cluster. The assumption clusters always have the required number of nodes, and they all have the same workload, makes the software architectures for validation simpler, although, in standard Globus distribution, a tool to accomplish the task, namely Glue (http://www.globus.org/mds/glueschema.html), is already available. On the other hand, a different solution to those problems would be to register a valid LDIF file in MDS with the information about the cluster we want to be advertised by the information system. In this way we can register, for example, the total number of nodes available in the cluster, the minimum and maximum memory and CPU utilization of nodes, which can be used to speed up the selection phase.

Selection algorithm, shown in **Algorithm 2,** returns either a machine that, given its actual status, can execute the application within *Application-time* (*found=true)* or the information no resource is available (*found=false*).

The input parameters are taken from the performance database and consist of the CPU frequency of cluster nodes on which the tests have been performed, their CPU utilization at the moment of tests, memory footprint of the job, number of nodes on which it has been processed and its execution time. For a dynamic property, such as CPU utilization, the geometric mean with respect to the last 15 minutes is considered, and its behavior will be monitored during application execution by the monitoring subsystem.

```
Procedure Selection( CPU_frequency,  CPU-percentage-usage, MEM-size-test, p, T_b (p),
                     Application-time, OS type, register)
found=false
selected=null
list=discovery(OS type, MEM-size-test, p, register)
while (next machine in list &&! found)

CPU_speed = MDS-Cpu-speedMHz
        CPU_free   = MDS-Cpu-Free-15minX100
        MEM_free   = Mds-Memory-Ram-freeMB
        if (MEM_free  >0 && CPU_free  >0) then
          Δ_CPU  =(2−CPU_free) CPU_speed/CPU_frequency
          O_MEM = (MEM_size_test -   MEM_free) / (MEM_size_test)
                  T_f (p)= T_b (p) *(α*Δ_CPU +β*O_MEM )
        end if
        if (T_f (p) + WT(p) < Application-time) then
        found=true
        selected=machine
      end if
end while
return selected
```

Algorithm 2. Selection

The other parameters are the application time, which is the maximum wall-clock time the user can wait, the operating system required and the register name. The algorithm uses the long term scheduling expected waiting time $WT(p)$ to take into account the time the job will be in the queue before being executed.

The selection algorithm currently uses a greedy algorithm to choose the first machine that can complete the job within the application time request. This strategy allows the selection of a resource without an exhaustive search. Other selection algorithms can be considered. For example, a *next-fit* algorithm could be implemented, in which the selection starts from the last selected resource and chooses the next available. Which approach is best will depend on the exact sequence of resource requests. The proposed algorithm is not only the simplest but usually the best and the fastest as well. On the other hand, the next-fit tends to discover machines at the end of the list, which produces a better workload balance. In the algorithm, new performance figures are not incorporated in the database, to be taken into account for following runs. The latter strategy could provide better estimates, and, in order to limit the size of the database, an exponential mean of the results could be stored.

The performance model takes into account the timing of a previous execution of the application, together with the characteristics of the cluster CPUs on which it run and the memory footprint of the application:

$$T_f(p) = T_b(p) * (\alpha * \Delta_{CPU} + \beta * O_{MEM}),$$

which consists of two terms, one related to the CPU effects, the other to the memory ones. If the performance database contains a run which refers to processors with

different CPU clock, the information is processed and the new execution time is supposed to be proportional to:

$$\Delta_{CPU} = (2\text{-}CPU\_free)\ CPU\_\ speed/CPU\_frequency,$$

where the numerator represents the speeding up factor of the discovered CPU, with respect to its workload, and the denominator the one in the performance database.

Current selection phase assumes, as the discovery one, that all CPUs are from the same processor family. Furthermore, it is supposed performance information contained in the database has been obtained on a machine with sufficient memory.

Without a performance model of the application, taking into account its communication and computational complexity, it is not possible to forecast how a variation in CPU usage, network speed or latency can affect application performance at runtime. The present broker selects the most promising resources among the available ones. The runtime analysis of performance achieved by application is demanded to monitoring subsystem, as it will be seen in the sequel.

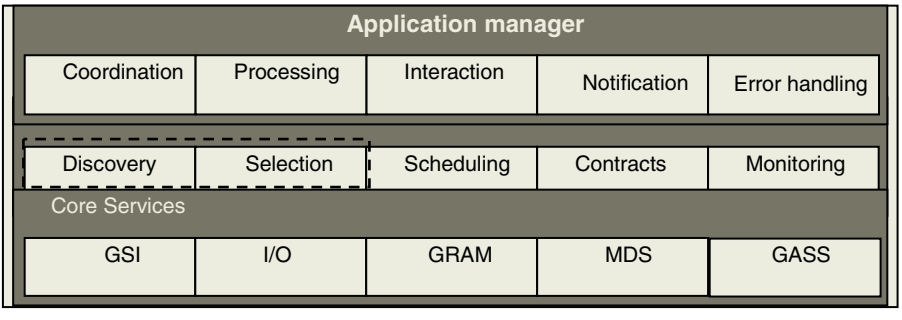


Fig. 2. Software architecture of MedIGrid

Software architecture of MedIGrid is composed of three layers: *core services*, which is based on Globus toolkit, *collective services*, including the resource broker, and *application manager*, in which there are all application specific software components. Discovery algorithm in MedIBroker uses security modules (GSI) to access the information system (MDS) and provides information to Globus resource allocation manager (GRAM), regarding available computing resources. It uses performance statistics (Performance) together with user's and reconstruction modules requirements to select available ones. MedIGrid application manager instantiates requests to underlying layer components in order to run user's jobs, which are submitted via a user portal and post-processed with a set of plug-ins [4]. **Figure 2** shows all such software compponents organized into layers and the resource broker components are inside the dotted line.

4 Experiments and Concluding Remarks

The experiments have been designed to assess the validity of the model for performance prediction, in case of variable processor utilization; we decided to measure the execution time in an environment in which the workload could be injected in a controlled way. *Overload* software package has been implemented to load a multicom-

puter with a given percentage of CPU utilization for a given period of time. The program is also capable to maintain a specified main memory utilization, but this feature has not been used, since the memory footprint of the reconstruction program has never exceeded 1MB.

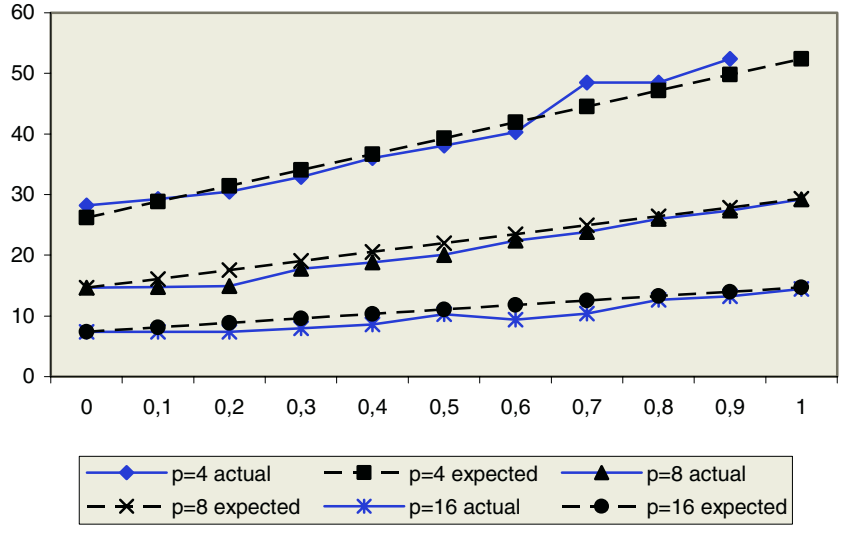


Fig. 3. Execution times in second

Tests have been performed on a Beowulf cluster, located in Naples at ICAR-CNR, consisting of 16 Pentium 4 1.5 GHz, with 512MB RAM, connected with a Fast Ethernet network. Each node runs a RedHat 9 distribution, Linux kernel 2.4.20, gcc compiler 2.96, mpich 1.2.5; MedIBroker has been implemented on Globus 2.4. Tests have been run on idle workstations during daily operation time; timing, in seconds, refers to wall clock time.

Discovery and selection procedures are implemented with bash scripts and C programs. Measurements refer to the execution time of MedITomo [3], the parallel software for 2D+1 reconstruction of SPECT images. It is based on a conjugate gradient computational kernel, which is executed on each slice of the data volume in sequential and results are updated via a collective gather operation. Then, the kernel has a fixed execution profile in the sense that the amount of data is the same for each job instance. **Figure 3** shows the execution time of MedITomo on $P = 4, 8$ and 16 processors, when CPU utilization increases from 0 to 100%. Results show the expected execution are predicted with a relative error of less then 0.1.

In conclusion, first results have been presented, and more testing is needed to completely validate the performance prediction model. Further, the model does not describe different network technologies and variable traffic. The broker only selects nodes within a single cluster. In future it will be investigated how MedIBroker can be recruit distributed computing elements for a really distributed computation. This requires a major shift in reconstruction algorithms and techniques, which need to be

tailored for the different computing paradigm. Nevertheless, the availability of such a tool would greatly influence the exploitation of new algorithms, since it would provide an easy to use environment in which it is possible to test new algorithms.

References

[1] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, C. Fabianek, T. Hiroyasu, E. Meek, M. Miller, K. Sagi, K. Seymour, Z. Shi, and S. Vadhiyar Users' Guide to NetSolve V2.0, 2002.

[2] M. Bertero, P. Bonetto, L. Carracciuolo, L. D'Amore, A. Formiconi, M. R. Guarracino, G. Laccetti, A. Murli and G. Oliva, "MedIGrid: a Medical Imaging application for computational Grids", in Proceedings of IPDPS 2003, IEEE Computer Society Press, April 2003.

[3] P. Boccacci, P. Calvini, L. Carracciuolo, L. D'Amore, and A. Murli, "Parallel Software for 3D SPECT imaging based on the 2D + 1 approximation of collimator blur", Ann. Univ. Ferrara, Sez. VII, Sc. Mat., vol. XLV, 1-0, 2000.

[4] P. Bonetto, G. Comis, A.R. Formiconi, and M.R. Guarracino, "A new approach to brain imaging, based on an open and distributed environment, in Proceedings of 1st International IEEE EMBS Conference on Neural Engineering, March 2003.

[5] P. Caruso, G. Laccetti and M. Lapegna, "A Performance Contract System in a Grid Enabling, Component Based Programming Environment", this volume.

[6] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid", in Proceedings of SC 2000, November 2000.

[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, "A Resource Management Architecture for Metacomputing Systems", in Proceedings of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.

[8] H. Dail, O. Sievert, F. Berman, H. Casanova, A. YarKhan, S. Vadhiyar, J. Dongarra, C. Liu, L. Yang, D. Angulo, and I. Foster, Scheduling in the Grid Application Development Software Project, Resource Management in the Grid, Kluwer, 2003.

[9] J. Frey, T. Tanenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", in Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10), August, 2001.

[10] W. Lee, S. McGough, S. Newhouse, and J. Darlington, Load-balancing EU-DataGrid Resource Brokers, Proceedings of UK e-Science All Hands Meeting 2003.

[11] H Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova, "GridRPC: A Remote Procedure Call API for Grid Computing", Submitted to the Workshop on Grid Computing, Baltimore, MD, 18th November 2002.

[12] M. Roehrig W. Ziegler, and P. Wieder, "Grid Scheduling Dictionary of Terms and Keywords", Grid Scheduling Dictionary WG, Global Grid Forum, November 2002.

[13] S. Vadhiyar, and J. Dongarra, "A meta-scheduler for the Grid", in Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing, July 2002.

[14] S. Vadhiyar, and J. Dongarra, "GrADSolve—a grid-based RPC system for parallel computing with application-level scheduling", in Journal of Parallel and Distributed Computing, v. 64, n. 6, pp 774-783, 2004.

[15] Weissman, J. Prophet, "Automated scheduling of SPMD programs in workstation networks", Concurrency: Practice and Experience, vol. 11, n. 6, 1999.

A Performance Contract System in a Grid Enabling, Component Based Programming Environment*

Pasquale Caruso[1], Giuliano Laccetti[2], and Marco Lapegna[2]

[1] Institute of High Performance Computing and Networking, Naples branch,
National Research Council – via Cintia Monte S. Angelo, 80126 Naples, Italy
[2] Department of Mathematics and Applications,
University of Naples Federico II – via Cintia Monte S. Angelo, 80126 Naples, Italy

Abstract. In these years, grid computing is probably the most promising approach for building large scale and cost effective applications. However, this very popular approach needs a sophisticated software infrastructure to address several requirements. One of these requirements is the ability to sustain a predictable performance in front to the fluctuations related to the dynamic nature of a grid. In this paper we describe design and realization of a Performance Contract System, a software infrastructure that manages the computational kernel of a grid application with the aim to face such aspect of the grid computing, as well as the strategies and the experiences to integrate it in a grid-enabling component-based programming environment still under development.

1 Introduction

As stated in [7], a Grid is a system that "… coordinates resources that are not subject to centralized control, … using standard, open, general purpose protocols and interfaces,… to deliver non trivial qualities of services". That means that a Grid infrastructure is built on the top of a collection of disparate and distributed resources (computers, databases, network, software …) with functionalities greater than the simple sum of those addends [8]. The "added value" is a software architecture aimed to deliver good Quality of Service (QoS), so a stronger attention has been recently given on the technologies enabling it (see for example [10]). Inside this software infrastructure, a significant part, known as Performance Contracts System, is devoted to the aspects related to the response time and to the delivered performance. Grid topics related to performance contract systems have been widely studied in the last years, mainly in the GraDS project (see for example [1,2]). Other papers (see [15]) report studies about the forecast of the performances in distributed computing environment, by using algorithms simulating an ideal customer, opportunely defined by means of some rules of behavior, in terms of use of the resources. A Performance

* This work has been partially supported by Italian Ministry of Education, University and Research (MIUR) within the activities of the WP9 workpackage *"Grid Enabled Scientific Libraries"* , coordinated by A. Murli, part of the MIUR FIRB RBNE01KNFP *Grid.it* project *"Enabling Platforms for High-Performance Computational Grids Oriented to Scalable Virtual Organizations"*.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 982 – 992, 2005.

Contract of an application for a computational grid by means of the computational cost of the algorithm is defined in [19] and then it is checked and validated.

Approaches that make use of statistical data , on the other hand, are introduced in [13,22]. Results related to the development of performance contracts based on the fitting of runtime data are reported, finally, in [16]. With regard to the run time monitoring, several tools for distributed applications are available and they will be shortly described in Section 4 [18,20,27]. A statistical analysis that, instead of checking all the software modules of the application, uses only some meaningful sections of the application itself, is developed in [17].

This paper is therefore organized as follows: in Section 2 we outline our Performance Contracts System and its role in a grid application; in Section 3 we introduce the software environment in which the Performance Contract System will be integrated; finally, in Section 4, we show some computational results about the definition of the performance contract and the related monitoring of a parallel routine that is part of a medical imaging application.

2 The Role of a Performance Contract System in a Grid Application

One of the aspects of grid computing is the simple and transparent use of the computational resources of a distributed system [8]. To such aim it is "mandatory", in some way, the presence of several software units, that are side by side to the application. Among the tasks of such software modules there are, for example, the selection of the resources, the development of the performance contract, its monitoring and the management of possible violations of the contract itself. The module that manages all activities is the Application Manager (AM), whose outline (or workflow) is depicted in Fig. 1. Its main software component are:

1. Resource broker. This component selects the computational resources on the basis of information about the application (e.g. the dimension of the problem), the user (e.g. the time to solution, that is the maximum amount of time to complete the application), the state of the grid (e.g. resources available in that moment) and finally, information about previous executions (e.g. performances caught up on a machine already used). See [6,14] for an example of selection of the computational resources.
2. Contract developer. This component has in charge the definition of the Performance Contract on the basis of the resources selected in step 1 without other input from the user. These information are combined with those related to the computational features of the application (e.g. the computational cost) as well as to the performance of previous executions (e.g. stored in a "historical performance database"); more details related to these aspects are reported in the sequel.
3. Monitor of the application. This is a key software item for a reliable grid-enabled application, because the actual performance can be very different from that one specified in the Performance Contract. The dynamic nature of distributed resources not under the same centralized control, can do these values very different among them. Some existing tool for the monitoring of distributed application are shortly described in the next Section.

4. Manager of the violation policy. This is the software item aimed to take the suitable actions in case of violation of the Performance Contract. Typical actions are the migration of the application on other resources, redefinition of the terms of the contract or the addition of other computational resources. See [23] for an example of migration strategy.

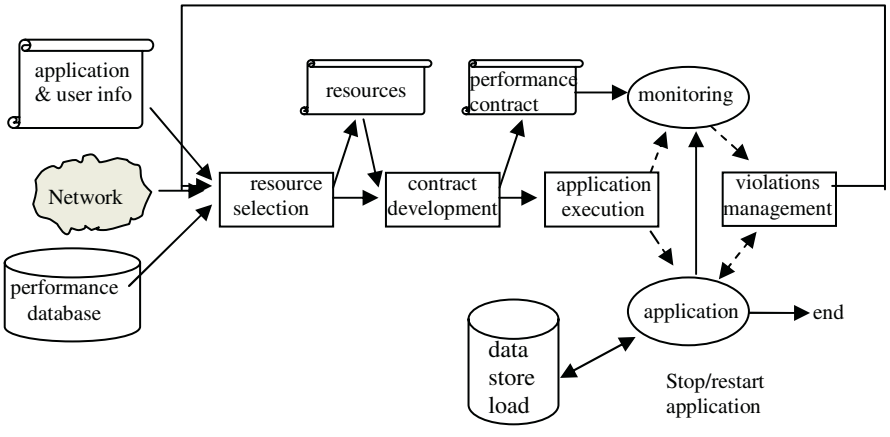


Fig. 1. Workflow of the Application Manager

A Performance Contracts System is the set of all software units of the Application Manager related to the performance contract and its monitoring. The definition of a Performance Contract is not a new one (see for example [26]), but in a grid environment it assumes a key role. A performance contract can be defined as a forecast of the performances of an application on given computational resource. More precisely, assigned:

- some computational resources (e.g. processors, memories, networks...)
- with given capability (e.g. computation speed, memory bandwidth...)
- and an application with given characteristics (e.g. dimension of the problem, amount of I/O, number of operations..)

a Performance Contract states

- the achievement of a fixed performances

There are several way to express a Performance Contract depending on the kind of application. Typical examples are the attainment of F operations/sec, the execution of I iterations/sec, transfer of B bytes/sec or the time necessary to compute a computational step (e.g. one frame in a image reconstruction problem). Obviously the choice among them depends on the features of the application.

Once selected the computational resources, the definition of the performance contract is essentially based on the following information:

1. use of a performance model based on the features of the application and of the selected computational resources. To be realistic, the definition of the model must take into account the computational cost of the algorithm, as

well as the workload of the resources, the fraction of peak performance actually obtainable, values of benchmarks, and so on. Such approach can be defined *Performance Model Approach*, and it is used, for example, in [19].

2. use of data related to the performances of previous executions. As an example, it is possible to use a database in which, for every computational resource selected in the time, average performances actually obtained, and the standard deviation (that can be used as estimate of the eventual deviation from the average value), are stored. The described approach can be defined *Historical Approach Performance*, and it is used, for example, in [22].

As previously said, because of the dynamic nature of a computational grid, during the execution of the application it is necessary to periodically check the actually obtained performances in order to compare them with those ones stated in the contract. Such monitoring is carried out by means of a suitable tool defined as *process monitor*: a sufficiently frequent check allows to take suitable actions (e.g.. migrating from a resource overloaded to another one, with the definition of a new contract; adding new computing resources, …). Tools like Automated Instrumentation and Monitoring System (AIMS) [27], Autopilot [20], Paradyn [18] or the commercial tool Vampir make this control. We note that all of them are based on the concept of *instrumentation* of the code, that is on the insertion of calls to library functions able to capture given information from the running code and to transmit them to a process monitor or a visualization tool.

3 A Grid-Oriented Component Based Programming Environment

Component programming model is a well known paradigm to develop applications. This approach, that can be considered as an evolution of the object-oriented model, that allow to build applications by binding independent software modules (the components) that interact with other components means of well defined interfaces (the ports) according a set of specific rules of the programming environment (the framework). The separation of the support code implemented into the framework from the application code into the components allows to the user to focus the attention on the application, avoiding to deal with environment dependent details [5].

Because the components describing the application can be implemented onto separate hardware and software environments, the component programming model is also a very promising approach to develop grid oriented programming environments [11,12]. So a new grid oriented component based programming environment is one of the goals of *Grid.it* Italian research project [24], where we are currently working to integrate a Performance Contracts System into the programming environment.

As already mentioned, the key role of the framework is to shade the details of the programming environment, by exposing only the services required to implement the components. In a grid oriented programming environment, therefore, beyond to the classical services related to the cycle life of the components (instantiation, resource allocation, …), the framework has to provide more sophisticated grid oriented services like resources discovery, remote data access as well the actions concerning the application structuring and rescheduling. As already said in Section 2, in a grid enabled application, these services are in charge of the Application Manager, that in

this context has a natural implementation in the framework. It is important to note that the middleware Globus [9] will be integrated in the framework in order to address the problems related to the access of the geographically distributed resources. In the *Grid.it* programming model, the components are supplied with several types of ports [24]:

1. Remote Procedure Calls (RPC) interfaces. These are the classical CCA-like ports mainly for client-server applications [5]. These interfaces define the kind of services that the component provides or uses.
2. Stream interfaces. This kind of interfaces allows the unidirectional communication of a data stream between two components. This kind of interfaces allows a better use of the bandwidth in case of high latency networks.
3. Events interfaces. These interfaces are used for the interaction of the components with the framework. The asynchronous events of a computational grid (e.g. the failure of resources) can be communicated to the components through these interfaces in order to take eventual actions for the rescheduling of the application on different resources.
4. Configuration interfaces. These interfaces allow to the Application Manager to access and to modify information and data inside the components and can be used for the reconfiguration steps (e.g. stop and restart of the application on other resources).

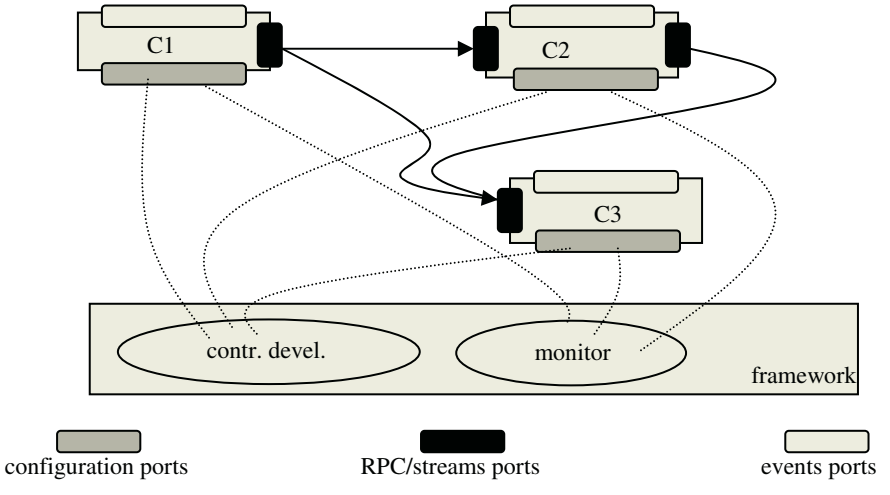


Fig. 2. Integration of the Performance Contract System in to the *Grid.it* environment

In order to integrate the Performance Contracts System, described in Section 2, into the *Grid.it* programming environment we firstly note that, while the application can be developed assembling the components directly by using the RPC or the streams interfaces, the software units composing the Performance Contract Systems (monitor and contract developer) and all related files and data structure can be di-

rectly implemented in the framework interacting with the application through the configuration interfaces

In Fig. 2 is shown an example of application with three components (namely C1, C2 and C3). The components exchanges their data by means of the streams port (black line) while the monitor and the contract developer access the data into the components by means the configuration ports (dotted lines).

More precisely, referring to the integration of the monitor in the environment, it is possible to add the components with proper scripting annotation, specific for the application (e.g. number of floating point operations in each iteration, number of communications,…), reporting which data have to be monitored. These information are accessed by the monitor through the configuration ports and are combined with the information directly acquired from the middleware implemented in the framework (e.g. number of processors to be use, kind of networks,…) and/or from the performances database, in order to define the Performance Contract. Through the same ports, the components provide to the monitor the run time values of the data to be monitored, in order to realize the monitoring process. In such a way the monitor can be based on a general purpose and application-independent template depicted in Fig. 3. An analogous approach can be used for the Performance Contract Developer.

- *Acquire from the components the data to be monitored through the configuration interfaces*
- *Acquire from the middleware the features of the resources to be use*
- *Acquire from the Performance Contract the values to be monitored*
- *Define the step time to get run time data from the components*
- *For each step time*
  - *Get run time values of the data to be monitored through the configuration interfaces*
  - *Test the values with the Performance Contract*
  - *if violation occurs apply violation policies*
- *endfor*

Fig. 3. Template for a general purpose monitor for grid applications

4 Computational Experiments

Our experiments were carried out on a preliminary version (ASSIST-CL 1.2) of the Grid.it environment [25]. The ASSIST programming model is based on a combination of the concepts of structured parallel programming and component-based programming. An ASSIST program is structured as a graph, where the nodes are the components and the edges are the component abstract interfaces, defined in terms of typed I/O streams. The basic unit of an ASSIST program is a component named *parmod* (parallel module), which allows to represent different forms of parallel computation. The user interface of the ASSIST environment is a coordination language, named *ASSIST-cl*.

A layered software architecture has been implemented to support the above programming model on the target hardware architectures, including SMPs, MPPs, and NOWs. An ASSIST-cl code is compiled and then it is loaded and run onto the target

architecture by a Coordination *Language Abstract Machine (CLAM)*. The CLAM is decoupled from the target hardware by a run-time support, named *Hardware Abstraction Layer Interface (HALI)*, which currently exports functionalities from the underling software layers. The ASSIST compiler translates ASSIST-cl source code into C++/HALI processes and threads, using predefined implementation templates. In running this code, the CLAM uses all the facilities provided by HALI, making no assumptions on the existence of other software running on the same nodes and competing to use the same resources. Finally the ACE library supplies standard routines to exchange data between processing elements with different architectures [21]. This is the layer of software that will be substituted with the middleware Globus in the Grid.it programming environment (see also the following Fig. 4).

To monitor the contract, we used the Autopilot library . This is a software environment for the adaptive run time control of geographically distributed applications. Such package is constituted by software items that allow to communicate data of programs in execution to a process monitor. Such software items are said *sensors*. Usually the sensors are used to capture the data related to the effective performance of the computational kernel to be monitored, in order to compare them with those stated in the Performance Contract. Further it is possible to use separate sensors in each process of a distributed application, so that the monitor is able to determine exactly which component of the application eventually causes the violation. Further, Autopilot is able to modify the value of variables of the executing applications, by means of the so-called *actuator* : the presence of the actuators is fundamental for example in a migration step. It is important to note also that Autopilot library does not introduce significant overhead in the software environment [20] and it uses the same Globus middleware that will be used in the Grid.it environment. In the following Fig. 4 the software architecture to realize our experiments is shown. In such an architecture it should be noted the role of the Autopilot library used to realize the monitor process, in accordance with Fig. 2.

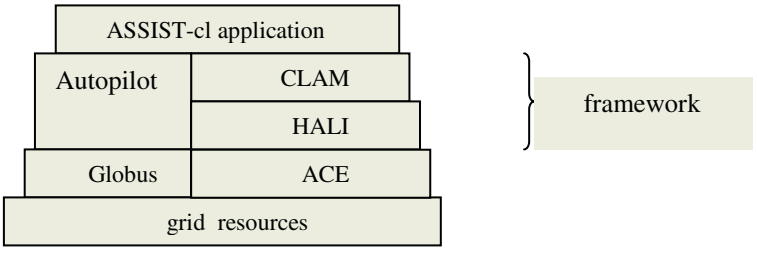


Fig. 4. Software architecture of ASSIST whit the Autopilot library

The computational kernel we used for our tests is based on the Coniugate Gradient (CG) algorithm, implemented in a routine of the the parallel library Meditomo realized to be used in the medical imaging application MediGrid [3,4], that reconstructs 64 independent bi-dimensional images,using 10 iterations of CG for every image, for a total of 640 iterations. Features of the matrices involved are: sparsity, not structure ness, order $n = 10^3$. For this problem we developed a parmod for the re-

construction of the 64 images, where the workload among the processors is distributed dynamically by using a *farm:* a parallel construct available in ASSIST. With this construct, each of the 64 images appearing on the input stream of the parmod is processed, independently from the other ones, by the first free processor.

As previously mentioned, and following a consolidate way, to define a contract it is necessary to know something about the past, in the sense of a historical database containing info regarding performances of previous executions.

Table 1. executions time for the reconstruction of the 64 images

	P=1	*P=2*	*P=4*	*P=8*	*P=12*
exec time without I/O (T_P)	2494	1261	629	313	234

Table 1 shows a very simple example of record of such database, reporting the execution time T_P, in seconds on P processors, of the computational kernel on a dedicated Linux Beowulf cluster with 12 Pentium 2 processors running at 550 MHz, connected by a Fast Ethernet switch at 100 Mbit/sec. We emphasize that such a times does not consider the I/O phases before and after each conjugate gradient. Such a values confirm however the natural parallelism of our problem, because we found that $T_1 \cong P \cdot T_P$.

On the basis of these data, and referring to the definition given in Section 2, we can define the Performance Contract as follow:

- given P processors (the computational resources)
- able to execute the given application in *2494/P* secs (the capability of the resources)
- and an application based on 640 iterations of the Coniugate Gradient
- the Performance Contract, expressed in term of seconds for one iteration, establishes that
- one iteration of the conjugated gradient has to be be executed in *2494/640=3.9* seconds independently from the number of processors *P*.

The monitored data are those ones defined in the Performance Contract, that is the execution time (Wall Clock Time) of one iteration of the conjugated gradient. By integrating the Autopilot sensors in the routine, we were able to carry out a set of experiments on the Beowulf machine, accessing runtime by means of the monitor, the Wall Clock Time of the execution of one iteration of the conjugated gradient every 40 seconds. After 150 seconds, one of the four processors (Node 1) has been overloaded by a process, stranger to the application, that engages the CPU for approximately 120 seconds before ending. Such overload is aimed to simulate the dynamical nature of the computational environment, in order to check if the performance contracts system, in this case, is able to finding the violation of the contract.

In Figure 5 the results of our test are reported, where on the x-axis is reported the time and on the y-axis is reported the execution time for one iteration of the Conjugate Gradient as caught by the monitor in 4 processors. Further is reported the value of the Performance Contract (PC). It can be view that, when the nodes of the Beowulf are not overloaded with other applications, the monitored values of the execu-

tion time agree with those stated in the Performance Contract. Moreover it is possible to observe that when the Node1 is overloaded with other applications, the actual value of the execution time is very different from that received from the monitor. Such first experiments confirm that our performance contracts system is able to define a realistic contract, able to preview the performance of the application in normal situation, and also to find violations of the contract itself. Such results are encouraging for future developments of the aspects related to the performance contracts system, as for example, the definition and implementation of suitable strategies to face violations of the contract itself.

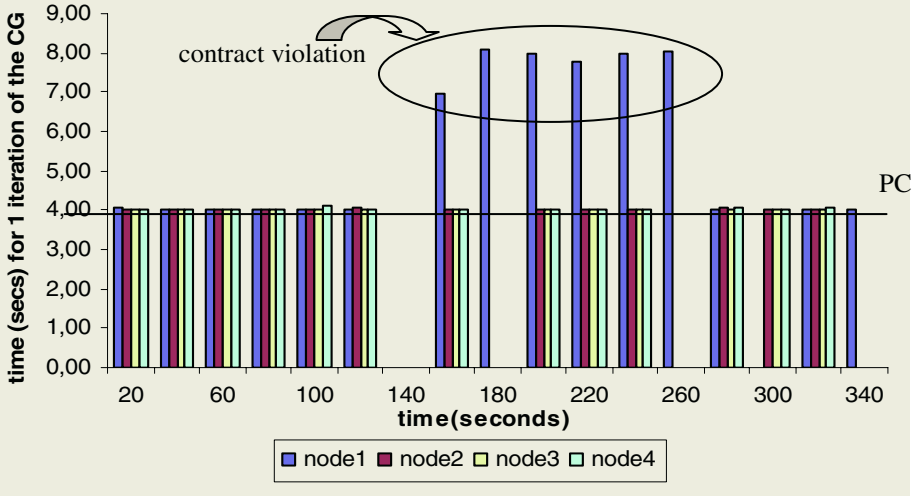


Fig. 5. Results of the monitoring process

References

1. R. Aydt, C. Mendes, D. Reed, F. Vraalsen - Specifyng and Monitoring GRADS contracts - available to the URL http://hipersoft.cs.rice.edu/grads/publications/grid2001.pdf
2. F. Berman, To Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnson, K. Kennedy, C. Kasselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolsky - The Grads Project: Software support for High Performance Grid Applications - *Int. Journal on High Performance Applications*. Vol 15 (2001), pp. 327-344.
3. M. Bertero,P.Bonetto, L.Carracciuolo, L.D'Amore, A.Formiconi, M.Guarracino, G.Laccetti, A.Murli, and G.Oliva – A Grid-Based RPC System for Medical Imaging - Parallel and Distributed Scientific and Engineering Computing: Practice and Experiences, Advances in Computation: Theory and Practice, vol. 15, Y.Pan and L.Yang (eds.), 2004, pp. 189-204.
4. P. Boccacci, P. Calvini, L. Carracciuolo, L, D'Amore, A. Murli - Parallel Software for 3D SPECT imaging based on the 2D + 1 approximation of collimator blur – *Ann. Univ. Ferrara, sez. VII, Sci. Mat. Vol. XLV, 2000*
5. CCA Forum Home page. http://www.cca-forum.org

6. K. Cooper et al. – New Grid Scheduling and Rescheduling Methods in GraDS Project – available at URL http://citeseer.ist.psu.edu/697420.html
7. I. Foster - What is the Grid? A three point checklist - available at URL http://www-fp.mcs.anl.gov/~foster/Article/WhatIsTheGrid.pdf
8. I. Foster , C.Kesselman - The Grid: Blueprint for a New Computing Infrastructure - Morgan and Kaufman 1998
9. I. Foster , C.Kesselman - Globus: a metacomputing infrastructure toolkit - *Int. Journal on Supercomputing Application*, vol. 11 (1997), pp. 115-128
10. I. Foster, C. Kesselman, J. Nick, S. Tuecke – The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Global Grid Formum, 2002
11. N. Furmento, W. Lee, A. Mayer, S. Newhouse, J. Darlington - ICENI: An Open Grid Service Architecture Implemented with Jini – Supercomputing 2002
12. M. Govindaraju, S. Krishnan, K. Chiu, A. Slominski, D. Gannon, and R. Bramley : XCAT 2.0: A Component-Based Programming Model for Grid Web Services.. Technical Report-TR562, Department of Computer Science, Indiana University. Jun 2002.
13. J. Gehring, T. Reinefeld - MARS, framework for minimizing the job execution time in a metacomputing environment- *Future Generation Computer Systems*, vol. 12 (1996), pp. 87-99
14. M.R.Guarracino, G.Laccetti, A.Murli – Application Oriented Brokering in Madical Imaging: Algorithms and Software Architecture – this volume
15. N. Kapadia, J. Fortes, C. Brodley - Predictive Application Modeling Performance in a Computational Grid Environment - Eighth IEEE Int. Symp. On High Performance Distributed Computing (1999), pp. 47-54
16. C. Lu, D. Reed - Compact Application Signature for Parallel and Distributed Scientific Codes - *Proc. of Supercomputing 2002*, (SC2002), Baltimore
17. C. Mendes, D. Reed - Monitoring Large Systems via Statistical Sampling - Proc. LACSI Symposium, Fe Saint, 2002
18. B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Bruce Irvin, K. Karavanic, K. Kunchithapadam, T. Newhall - The Paradyn Parallel Performance Measurement Tools - *IEEE Computer* vol. 28 (1995) pp. 37-46
19. F. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, S. Vadhiyar - Numerical Libraries and the Grid: The GrADS Experiment with ScaLAPACK, - Technical report UT-CS-01-460, 2001
20. R. Ribler, J. Vetter, H. Simitci, D. Reed - Autopilot: Adaptive Control of Distributed Applications - *Proc. of High Performance Distributed Computing Conference*, 1998, pp. 172-179
21. D.C.Schmidt, T. Harrison, E. Al-Shaer – Object Oriented components for high speed network programming – in proc. of 1st conf. on OO technology and systems (1995)
22. W. Smith, I Foster V. Taylor. - Predicting application run times using historical information. - Proc. Of the IPPS/SPDP' 98 workshop on job scheduling strategies for parallel processing (1998)
23. S. Vadhiar and J. Dongarra – A performance oriented migration framework for the grid - Proceedings of the 3st International Symposium on Cluster Computing and the Grid, 2003
24. M. Vanneschi – High Performance Grid Programming Environments: The Grid.it ASSIST Approach , *invited lecture, ICCSA 2004.*
25. M. Vanneschi – The programming model of ASSIST, an environment for parallel and distributed portable applications – Parallel Computing, vol. 28 (2000), pp. 1709-1731

26. F.Vraalsen, R. Aydt, C. Mendes, D. Reed – Performance contracts: predicting and monitoring application behaviour – Proc. IEEE/ACM Second Intern. Workshop on Grid Computing, Denver, 2001, Springer Verlag LNCS, vol. 2242, pp. 154-165
27. J. C. Yan, M. Schmidt and C. Schulbach. "The Automated Instrumentation and Monitoring System (AIMS) -- Version 3.2 Users' Guide". *NAS Technical Report. NAS-97-001*, January 1999

A WSRF Based Shopping Cart System

Maozhen Li[1,2], Man Qi[3,2], Masoud Rozati[4], and Bin Yu[1]

[1] School of Engineering and Design,
Brunel University,
Uxbridge, UB8 3PH, UK
{Maozhen.Li, Bin.Yu}@brunel.ac.uk
[2] Dept. of Computer Science,
Guangxi University of Technology,
Liuzhou, Guangxi, 545006, P.R.China
[3] Dept of Computing,
Canterbury Christ Church University College,
Canterbury, Kent, CT1 1QU, UK
[4] ANSARI GmbH,
Friedrich-Ebert-Damm 160,
D-22047 Hamburg, Germany
rozati@gmx.net

Abstract. Web Services Resource Framework (WSRF) is a set of specifications that represents a convergence point of the Web services and the Grid services communities. This paper presents our early experience with WSRF. A shopping cart system has been implemented with WSRF supported Globus toolkit 3.9.2 (GT3.9.2). Based on the system, the performance of the WSRF core in GT3.9.2 has also been evaluated.

1 Introduction

Based on Web services [1], Open Grid Services Architecture (OGSA) [2] becomes the de facto standard for building services-oriented Grid systems [3]. In the context of OGSA, a Grid service is a Web service with the following major extensions:

- A Grid service can be a transient service that can be dynamically created and explicitly destroyed.
- A Grid service is a stateful service that is associated with service data.
- Clients can subscribe to Grid services for notifications of events of interest.

OGSA merely defines what interfaces are needed, but does not specify how these interfaces should be implemented. It is the Open Grid Service Infrastructure (OGSI) [4] that defines how to technically implement these interfaces.

However, the Web services community has recently criticized the work on the extension of standard Web services in OGSI mainly because the OGSI specification is too heavy with everything in one specification, and it does not work well with

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 993 – 1001, 2005.

existing Web services and XML tooling [5]. In January 2004, the Globus Alliance [6] and IBM in conjunction with HP introduced the WSRF [7] to resolve this issue.

WSRF represents a convergence point of the Web services (WS) and Grid services communities. It introduces the concept of WS-Resource to model Web services with stateful resources [8]. WSRF is a set of specifications of which the following ones have been drafted:

- *WS-ResourceLifetime* [9] defines mechanisms for service requestors to request Web services to destroy associated WS-Resources immediately or after certain time.
- *WS-ResourceProperties* [10] defines the means by which the definition of the properties of a WS-Resource may be declared as part of a Web service interface.
- *WS-Notification* [11] defines mechanisms for event subscription and notification using a topic-based publish/subscribe pattern.
- *WS-BaseFaults* [12] defines an XML Schema for base faults, along with rules to specify how these faults types are used and extended by Web services.
- *WS-ServiceGroup* [13] defines a means by which Web services and WS-Resources can be aggregated or grouped together.

WSRF has been receiving more and more attentions from both Web services and Grid services communities since its first draft drew up in Jan. 2004. Work to fully implement WSRF specifications is still ongoing. For example, it is expected that a WSRF based Globus toolkit 4 (GT4) [14] will be available in Jan. 2005.

In this paper, we present our experience with WSRF based on the implementation of a shopping cart system with GT3.9.2 [15], an early work of GT4. Based on the system, the performance of the WSRF core in GT3.9.2 has also been evaluated.

The remainder of the paper is organised as follows. Section 2 presents the shopping cart system. Section 3 evaluates the performance of WSRF from the aspects of WS-Resource creation/destruction, service invocation, and WS-Resource property access. The evaluation has been performed using one computer and using two computers connected by a local area network. Section 4 concludes the paper and gives a discussion on the reliability of GT3.9.2.

2 A WSRF Based Shopping Cart System

In this section, we present a shopping cart system implemented with GT3.9.2. This prototype system is used for the performance evaluation of the WSRF core in GT3.9.2.

The shopping cart system supports multiple clients to access services. As shown in Fig. 1, a client uses SOAP [16] to access a shopping cart service through its interface defined in WSDL [17]. The cart is a stateful WS-Resource. The main components implemented in the system are described below.

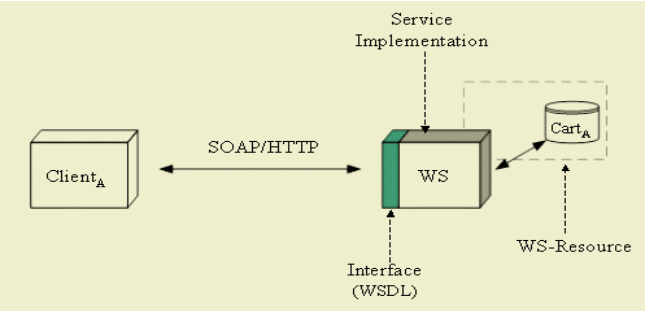


Fig. 1. A WSRF based shopping cart system

2.1 Cart

The Cart is a stateful WS-Resource that keeps the identifier of picked articles as well as the desired quantity of them. It has two properties: total price of the articles and the number of articles. We store the Cart and its properties in a file to make it a persistent WS-Resource (PersistentCart) in case of a failure. The cart ID, which is transparent to the client, is embedded in the WS-Resource qualified endpoint reference of the Cart and will be carried along in all message exchanges between the client and the server.

```
public class Cart
      implements ResourceLifetime, ResourceIdentifier, ResourceProperties,
TopicListAccessor
    {
        . . .
        protected Hashtable articles;
        protected ResourceProperty totalPrice;
        protected ResourceProperty articleCount;
        ...
    }
```

Fig. 2. The Cart class

As shown in Fig. 2, the Cart class implements *org.globus.wsrf.ResourceProperties* interface for an access of resource properties, *org.globus.wsrf.ResourceIdentifier* interface for resource identification, *org. globus.wsrf.ResourceLifeTime* interface for resource lifetime management. In addition, the Cart also implements *org.globus.wsrf.TopicListAccessor* interface for resource notification.

2.2 Cart Service

The Cart service (cartService) is the actual Web service component in the system. The *cartService* exposes two operations through its public interface:

- *addArticle*
- *removeArticle*

In addition, cartService provides two methods for creating and destroying the Cart WS-Resource and works as a factory service for the WS-Resource.

2.3 Cart Home

The stateful resources are deployed in GT3.9.2 container as Java Naming and Directory Interface (JNDI) [18] resources. A JNDI resource has a home class that is responsible for the creation and location of that resource. The Cart WS-Resource home class (CartHome) should implement the *org.globus.wsrf.impl.ResourceHomeImpl* interface. The *create()* method of the CartHome class is called whenever a request to create a new instance of this resource is received. Fig. 3 shows the sequences to create a resource.

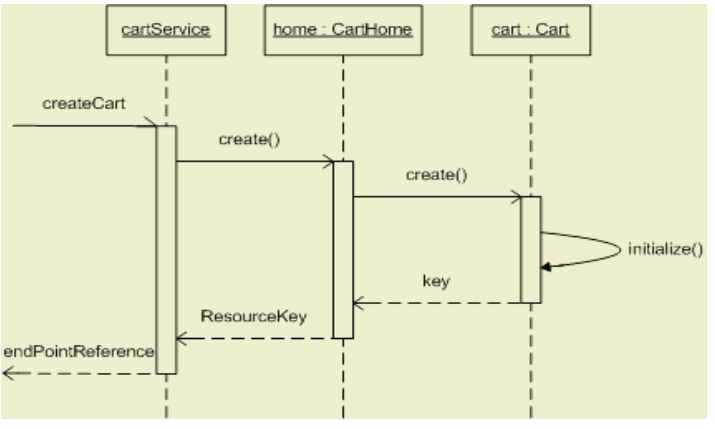


Fig. 3. Sequences to create a resource

2.4 Cart Client

A client uses the automatic generated stubs to locate the Cart Web service and send requests to create a WS-Resource and invoke operations upon it. GT 3.9.2 generates helper classes such as *ServiceNameAddressingLocator* that finds and returns the service proxy object using the service address (URI). In the shopping cart system, the CartPortType class is the proxy of cartService at the client side. Fig. 4 shows how a client uses *CartServiceAddressingLocator* to find the CartPortType and then request it to create a Cart WS-Resource.

To subscribe to a notification, the client should implement the *org.globus.wsrf.NotifyCallback* interface. The notification workflow at the client side is described as follows:

- The client receives an instance of *NotificationConsumerManager* and then invokes it. This object listens for incoming notifications.
- The client calls the *subscribe()* method of the Cart Web service. The subscription request contains the topic name of interest and the topic dialect the name uses. (GT 3.9.2 currently only supports *SimpleTopicDialect*)
- The public method *deliver()* will be called back whenever a notification is received.

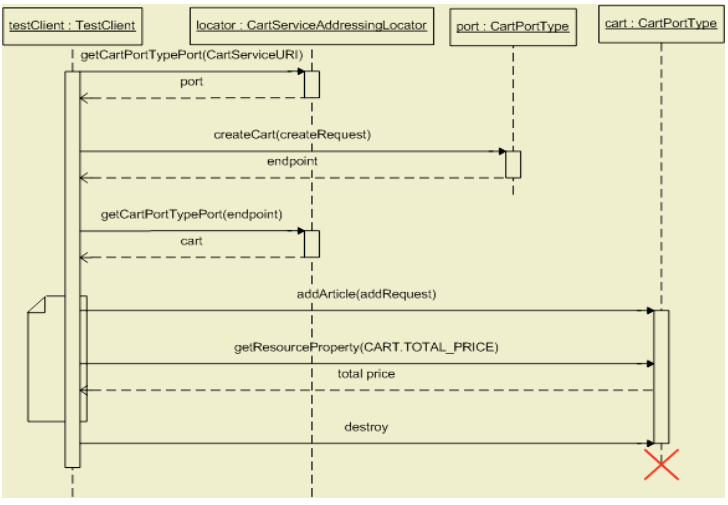


Fig. 4. Sequences for requesting a Cart service

3 Performance Evaluation

We have constructed two experimental environments for the evaluation of the WSRF based shopping cart system implemented with GT3.9.2. In the first environment, the Cart client and Cart service were deployed on the same computer (Intel Pentium III 850MHz, Windows 2000 Professional, 512 MB RAM). In the second environment, the Cart client and Cart service were deployed on two computers connected by a local area network with a bandwidth of 100Mbps. The client computer is an Intel Pentium III 550MHz with a memory of 256MB running Windows 2000 Professional. The server computer is an Intel Pentium III 850MHz with a memory of 512MB running Windows 2000 Professional. For our tests, we considered the single-server/multiple-clients situation as shown in Fig.5.

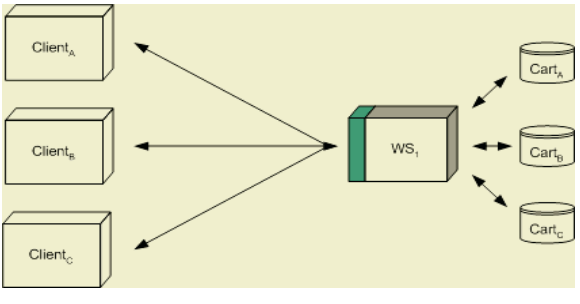


Fig. 5. Multiple clients access a single Cart service

The response time of the following four operations have been measured using variable number of clients:

- The creation of the Cart WS-Resource (CreateCart method)
- The destruction of the Cart WS-Resource (Destroy method)
- The invocation of the Cart service (addArticle method)
- The access of the Cart WS-Resource property (GetProperty method)

Clients can subscribe to the Cart service for notification of events of interest. The Cart service published two notification topics: ARTICLE_COUNT and TOTAL_PRICE. Clients who were interested in getting notified of changes of these values subscribed to these topics.

3.1 Performance Evaluation on One Computer Without Notifications

In this evaluation, the clients and the Cart service were deployed on one computer. The clients did not subscribe to the Cart service for any notifications.

As shown in Fig. 6, the creation of the Cart WS-Resource is the most time consuming step in the shopping cart system. This is because the initiations of WS-Resource need more time than other operations. Another reason is that we created a file for each Cart WS-Resource to make it persistent. When the number of clients tested is below 100, the time taken to invoke the Cart service and the time taken to access the WS-Resource property is roughly the same. However, as number of clients goes beyond 100, the time taken to invoke the Cart service is getting longer than that of the access of the WS-Resource property. Again, this can be explained due to file access in the AddArticle() method. The destruction of the Cart WS-Resource needs the least time among the four operations.

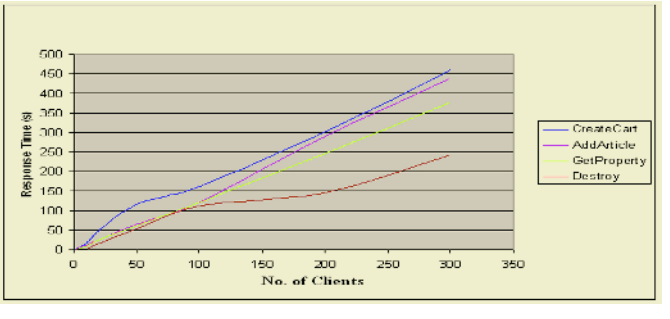


Fig. 6. Performance evaluation on one computer without notifications

3.2 Performance Evaluation on One Computer with Notifications

In this evaluation, the clients and the Cart service were deployed on one computer. The clients subscribed to the Cart service for notifications of events of interest.

As shown in Fig. 7, the use of notifications increases the overhead of the four operations. Again the creation of the cart WS-Resource is the most time-consuming operation among the four operations. However, compared with the evaluation de-

scribed in Section 3.1, the major difference here is that for a large number of clients, the destruction of the Cart WS-Resource needs significantly more time than before. Unlike previous tests without notification, we have experienced out of memory errors for 100+ clients. For example, in the test of 200 clients, 66% of the destruction requests were failed either due to the socket timeout or out of memory errors happened at the client side. Almost all tests with more than 200 clients in this evaluation were failed.

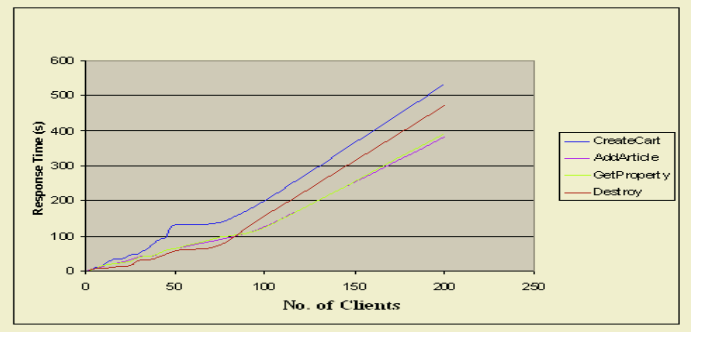


Fig. 7. Performance evaluation on one computer with notifications

3.3 Performance Evaluation in a LAN Without Notifications

In this evaluation, the clients and the Cart service were deployed on two computers connected by a 100Mbps local area network (LAN). The clients did not subscribe to the Cart service for any notifications.

As shown in Fig. 8, Similar to the evaluation described in Section 3.1, the time to create the Cart WS-Resource is the most time-consuming step among the four operations, and the destruction of the Cart WS-Resource needs the least time. However, compared with the evaluation described in Section 3.1, the time taken for each of the four operations performed in the LAN is less because the clients and the Car service were deployed on two computers using more resources than using only one computer.

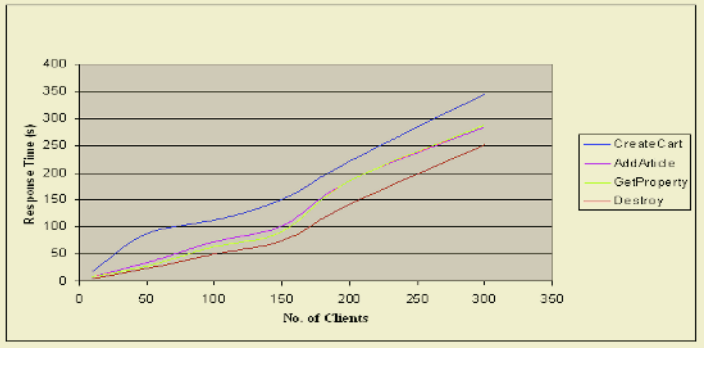


Fig. 8. Performance evaluation in a LAN without notifications

3.4 Performance Evaluation in a LAN with Notifications

In this evaluation, the clients and the Cart service were deployed on two computers connected by a 100Mbps local area network (LAN). The clients subscribed to the Cart service for notifications of events of interest.

As shown in Fig. 9, the use of notifications increases the overhead of the four operations. However, compared with the tests performed on one computer with notifications as described in Section 3.2, the overhead incurred by notifications in a LAN is less in each of the four operations.

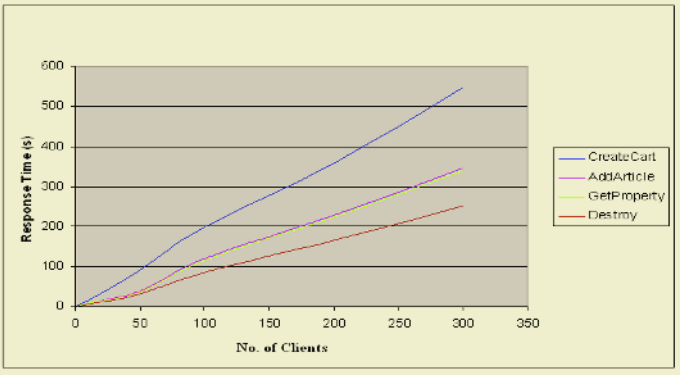


Fig. 9. Performance evaluation in a LAN with notifications

4 Conclusions

In this paper, we have presented a WSRF based shopping cart system implemented with GT3.9.2. Based on the system, we have evaluated the performance of WSRF in terms of WS-Resource creation and destruction, WS-Resource property access and Web services invocation. From the tests we know that, among the four operations, the creation of WS-Resource is the most time-consuming step, as it needs more time for initiations. Tests performed in a LAN have a better performance than that of the tests performed on one computer. The use of notifications increases the overhead of the four operations performed on one computer and in a LAN as well; however, the overhead incurred in a LAN is less than the overhead incurred on one computer as more computing resources are used.

The evaluation results have also shown a reliable container in GT3.9.2. Almost none of the tests performed at the server side crashed or incurred any errors. However, the client side incurred some errors when it had high load for opening connections for a large number of clients. When the number of clients went beyond 500, almost every destruction request ended up with a “socket read timeout” error at the client side. On the other hand, when the server had a high load, the notification callbacks were rarely received by clients even though they have subscribed to the events of interest.

References

1. Web Services, http://www.w3.org/2002/ws/
2. Foster, I., Kesselman, C., Nick, J., and Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, June 2002, http://www.globus.org/research/papers/ogsa.pdf
3. Foster, I. and Kesselman, C. (ed.): The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.
4. OGSI Working Group, http://www.Gridforum.org/ogsi-wg/
5. Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Maguire, T., Snelling, D., Tuecke, S.: From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution, Version 1.0, Feb. 2004. http://www-106.ibm.com/developerworks/library/ws-resource/gr-ogsitowsrf.html
6. Globus, http://www.globus.org
7. Czajkowski, K., Ferguson, D. F., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W.: The WS-Resource Framework, Version 1.0, March 2004, http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf
8. Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T., Vambenepe, W., Weerawarana, S.: Modelling Stateful Resources with Web Services, Version 1.1, March 2004. http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf
9. Frey, J., Graham, S., Czajkowski, K., Ferguson, D. F., Foster, I., Leymann, F., Maguire, T., Nagaratnam, N., Nally, M., Storey, T., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W., Weerawarana, S.: Web Services Resource Lifetime, Version 1.1, March 2004, http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf
10. Graham, S., Czajkowski, K., Ferguson, D. F., Foster, I., Frey, J., Leymann, F., Maguire, T., Nagaratnam, N., Nally, M., Storey, T., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W., Weerawarana, S.: Web Services Resource Properties, Version 1.1, March 2004, http://www-106.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf
11. Web Services Notification, http://www-106.ibm.com/developerworks/library/specification/ws-notification/
12. Tuecke, S., Czajkowski, K., Frey, J., Foster, I., Graham, S., Maguire, T., Sedukhin, I., Snelling, D., Vambenepe, W.: Web Services Base Faults, Version 1.0, March 2004, http://www-106.ibm.com/developerworks/library/ws-resource/ws-basefaults.pdf
13. Graham, S., Maguire, T., Frey, J., Nagaratnam, N., Sedukhin, I., Snelling, D., Czajkowski, K., Tuecke, S., Vambenepe, W.: WS-ServiceGroup Specification, Version 1.0, March 2004, http://www-106.ibm.com/developerworks/library/ws-resource/
14. GT4, http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/GT4Facts/index.html
15. GT3.9.2, http://www-unix.globus.org/toolkit/downloads/development/
16. SOAP, http://www.w3.org/TR/soap
17. WSDL, http://www.w3.org/TR/wsdl
18. JNDI, http://java.sun.com/products/jndi/

Grid Access Middleware for Handheld Devices

Saad Liaquat Kiani[1], Maria Riaz[1], Sungyoung Lee[1],
Taewoong Jeon[2], and Hagbae Kim[3]

[1] Computer Engineering Department, Kyung Hee University,
Giheung, Yongin, Gyeonggi 449-701, Korea
{saad, maria, sylee}@oslab.khu.ac.kr
[2] Department of Computer & Information Science, Korea University, Korea
jeon@selab.korea.ac.kr
[3] School of Electrical & Electronic Engineering, Yonsei University, Korea
hbkim@yonsei.ac.kr

Abstract. Grid technology attempts to support flexible, secure, coordinated information sharing among dynamic collections of individuals, institutions, and resources. The use of Grid services requires a resourceful workstation, specialized software installed locally and expert intervention. Mobile handheld devices in general do not posses enough computational and communication assets to meet the criteria for utilizing the Grid infrastructure services. We present the design of a middleware approach that aids handheld devices in this regard by wrapping the computational and resource intensive tasks in a surrogate and shifting them to a capable machine for execution[1]. Reduction in computational burden at the handheld device is analyzed in a test scenario.

1 Introduction

Grid [1] computing harnesses the abundant spare, and sometimes dedicated computational resources in a globally distributed computing environment and puts them to effective and optimal use. Grid setup is clearly a value addition to any organization, commercial or research. One of the main motivations behind the Grid infrastructure is to provide "a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities" [2]. Grid infrastructure has been put to use in areas like high energy physics [3], bio-medicine [4], aerospace and earth sciences, health-care [5] etc and is continuing to evolve and expand. Similar technology adoption trends are seen at the smaller scale. With ever decreasing costs and increasing functionality of small sized chips, mobile handheld devices e.g., Personal Digital Assistants (PDA) and smart phones, are becoming mainstream now. For a mobile user, a PDA takes place of his home/office PC while he is on the move; he can not only use the internet and check emails through wireless connectivity

[1] This work is supported in part by the Korea Ministry of Information and Communications' ITRC program in joint collaboration with ICU Korea.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1002–1011, 2005.

but can also write documents, play games, find street maps, make reservations at hotels and restaurants and perform similar utility tasks. A broad spectrum of internet services has become available for a mobile user. Grid and mobile computing however remain two disjoint phenomenons as yet, keeping users of both technologies from utilizing some propitious mutual benefits. While mobile elements will improve in absolute ability, they will always be resource-deprived relative to their static counterparts (desktops/workstations). In [6], the author argues that for a given cost and level of technology, considerations of weight, power, size and ergonomics will exact a penalty in computational resources such as processor speed, memory size, and disk capacity. These devices do not have enough resources in effect to utilize the Grid services comprehensively. The potential benefits of facilitating mobile devices in interacting with Grid services in the numerous fields are:

- Health care: A physician submitting digital charts to mammography Grid services [7] for analysis
- Emergency medical services: Submitting vital characteristics, medical history of a trauma patient to Grid services for identification
- Research: A physicist who needs to see graph plots of data produced as a result of high energy collisions between atoms and sub-particles on his PDA. The amount of information in data-stores, from which graphs are to be generated, will be in the range of several gigabytes or even tera bytes
- Weather: Forecasting and analyzing local weather conditions, storm formations while on the move
- Geology: Geologists sampling rocks and terrain and using handheld devices to submit data to Grid services for analysis

All these domains represent scenarios where a user wants to execute a resource intensive task at a location where computation resources are not available at hand. With ever increasing mobility of users and greater adoption of handheld devices, job submission to the Grid through handheld devices presents a viable solution for maximizing efficiency. Constraints that hinder handheld devices from such interactions include limited network bandwidth, CPU power, memory (small network buffers) and intermittent connectivity. Keeping the limitations in mind, we aim to define a middleware approach that will allow handheld devices, e.g. PDA units, to interact with Grid services while inducing minimal burden on the device itself. We demonstrate a solution based on Jini Network Technology's [8] Surrogate Architecture [9] which provides a network framework in which a device can deploy a client or a service on a device other than itself. Since we are stepping in a new realm of Grid access through handheld devices, many design and performance challenges need to be considered and countered. In the domain of Grid infrastructure, where services and data resources are replicated across geographical boundaries [10, 11], communication costs can be minimized by careful selection of intermediate network. The communication mechanisms involved in job submission, execution and resource access are optimized at three levels: 1) Selection of the host to which the device will submit the job/task for execution, 2) Resource access by the surrogate during execution and 3) filtering

and optimization of intermediate results that are to be transferred to the device from the remote machine. One possible approach for facilitating handheld device interaction with the Grid is to narrow down the criteria for Grid access and make it less resource hungry; but doing so will also take away several benefits. How can a resource constrained device be configured and supplemented with software based techniques to make it Grid-interaction capable? A handheld device wishing to host a service and unable to do so can be allowed to delegate this task to a relatively powerful machine (desktop, server). Conversely, if the interaction with remote Grid services proves too much for limited local resources of a handheld device, it can deploy the actual client functionality at an intermediate machine and receive the results in a form that is in keeping with its hardware resources. This second scenario has a greater probability of being used in real world applications and is the focus of our research. The 'service' or 'client' process, transferred from the device, is called a 'surrogate' (The term 'surrogate' is used to describe an entity that performs some action on behalf of another entity). The middleware component at intermediate machine, which provides the execution environment and access to extensive resources for the handheld device's surrogate, is called the 'Gateway Surrogate Host' or simply 'Host'. An interconnect mechanism, defined as "logical and physical connection between the surrogate host and a device" [12], also needs to exist. A handheld device that can communicate over IP (wireless or wired) can be programmed to shift its task processing to a host capable machine. An overview of our middleware approach is presented in Sect. 2. Section 3 deals with the communication mechanisms and the proposed optimizations in the middleware. Prototype implementation and test results are presented in Sect. 4. We conclude our discussion in Sect. 5 and also list relevant related work.

2 The Grid Access Middleware Architecture

A handheld mobile device having wired/wireless connectivity can utilize the functionality of its more capable computing peers for resource demanding tasks such as Grid service access, with the device itself only managing less intensive tasks like displaying the tailored results. The main concept driving our approach is to shift the 1) access to generic Grid services and 2) intensive task processing, from a resource constrained handheld device to a resource rich system (i.e. the Surrogate Host). This is to be achieved by wrapping the access and processing mechanisms in a 'surrogate' module and transferring to the host. Consider the example of a physicist provided in Sect. 1, where he needs to see graph plots, on his PDA, of data produced as a result of high energy collisions between atomic particles. The amount of information in data-stores from which graphs are to be generated will be in the range of several gigabytes or even tera bytes. Analysis of such data for the plotting of graphs is not a job for the handheld device. Moreover, the handheld device may have reduced network bandwidth, further diminishing the prospects of a successful remote analysis by the user. By utilizing the Jini Surrogate Architecture based middleware support, one can 'pack' the

functionality for data-stores' access mechanisms and graph plotting routines in a surrogate and transfer this surrogate to a host machine. The host machine will provide the surrogate with necessary resource rich execution environment and network connectivity. The surrogate is able to communicate back to the device (PDA) through available interconnects e.g. IP, USB, Bluetooth etc. In this way, the aforementioned tasks of service access and intensive processing can be shifted from the handheld device to a more appropriate host machine, with the device only managing less intensive tasks of displaying the tailored results returned by its surrogate. Figure 1 shows the middleware framework which consists of three distinct stacks deployed at the Gateway Surrogate Host, the Device and the surrogate. These are discussed one by one in the subsequent paragraphs.

2.1 Gateway Surrogate Host

Major technical hurdles make it impossible for Devices to exploit the benefits made available by the computational and data Grids, including the ability to execute applications whose computational requirements exceed local resources and reduction in job turn around time through workload balancing across multiple computational facilities. Gateway Surrogate Host is the middleware component that aids the Device to overcome these hurdles by accepting tasks, packed as surrogates, for execution. The middleware provided at these hosts consists of three main sub-modules. Host Adapter sub-module offers an interface to client devices for accessing the Gateway Surrogate Host. It enables the initial communication between the device and the host so that both can agree on the transfer of surrogate after authenticating the device and its corresponding surrogate. Once

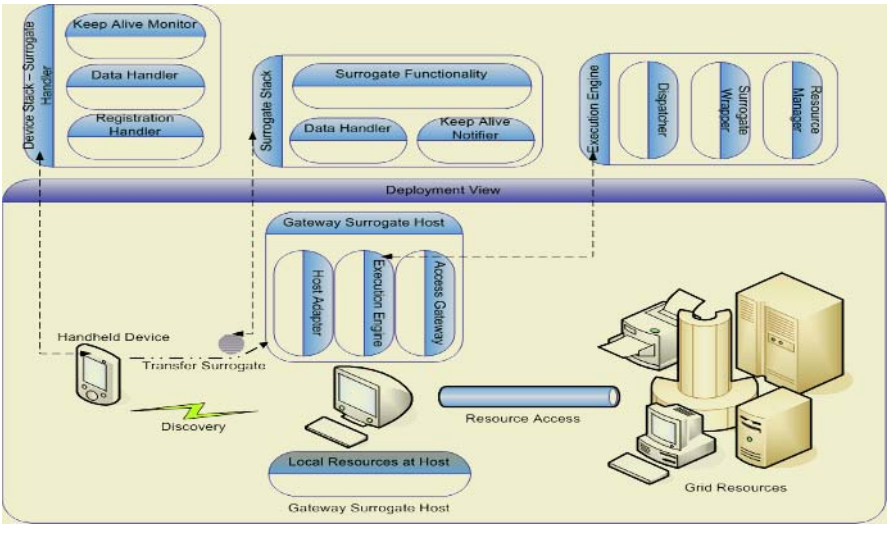


Fig. 1. Grid Access Middleware Stacks at the Device, Surrogate and Host

the surrogate is available at the host, it is delivered to the Execution Engine sub-module. It consists of a Surrogate Wrapper that exposes the functionality of the surrogate that is required to facilitate surrogate's execution at the host. Dispatcher allocates a separate thread for the execution of the surrogate from a thread pool and then activates the surrogate. Resources required for surrogates' execution are resolved and handled by the Resource Manager module. These include memory and disk space, JVM (form Java based surrogates, as is the case with our implementation), network resources etc. The Access Gateway sub-module provides interface to the external resources e.g. discovery of available Grid services and resources. A Gateway Surrogate Host announces relevant attributes including, but not restricted to:

- ID, Location, Currently hosted surrogates etc
- Network address and Discovery/Listening port for incoming Device/Client requests
- Available/Allocated Resources e.g. CPU, Memory, Storage, Network throughput
- Environment e.g. Java VM availability and version, SOAP/WSDL [13, 14], XML parser etc
- Grid services available through this Surrogate Host
- Proximity to service and client side

Advertising these attributes allows clients to locate appropriate hosts based on their location, network proximity and other desired features. This is further elaborated in section 3.1. Administrator of a host can restrict the number of surrogates that are allowed to execute, restrict memory, bandwidth allocation etc on per surrogate basis. Security policies can be configured based on public/private key pairs and digital certificates. The Gateway Surrogate Host is an extension of the basic Surrogate Host with added functionality for Grid access through the Access Gateway. It overcomes the major technical hurdles that keep the Devices from exploiting the benefits made available by the computational and data Grids [13, 14] by providing an interface to the Devices on one hand and to the Grid services on the other.

2.2 Device Stack

At the Device, a lightweight middleware stack is provided for facilitating coordination with its exported surrogate. The stack consists of a Surrogate Handler module which has three sub modules for providing services complementary to the middleware at the Gateway Surrogate Host. Registration Handler discovers, selects and registers with the Host, and transfers the surrogate. Once the surrogate is transferred, Keep Alive Monitor keeps track of the status of the surrogate. Data Handler retrieves the results from the surrogate-side corresponding module, and makes them available to the application executing at the device. Surrogate to be transferred can be stored at the Device itself or at a URL accessible store e.g. a web server or an FTP server.

2.3 Generic Surrogate

A generic surrogate for Grid service access contains the following features:

- Client authentication based on public/private key pairs
- Generic functionality to communicate and interact using WSDL/SOAP for web service based Grid services
- Persistency safe i.e. to be put to persistent storage if its functionality is periodic
- Migration - To be able to stop and save current execution, mark restore points and migrate to a different Surrogate Host

Functionality of the generic surrogate is incorporated at the top layer of the surrogate stack as shown in Figure 1, along with the specific logic of the extended Surrogate. Moreover, the surrogate has complementary modules for communicating with the middleware stack at the Device. The downloadable Surrogate can be located in the file system of the Device or at a URL accessible store e.g. a web server or FTP server. Some clients may be void of any Surrogates. These clients/devices are still able to use other deployed surrogates if they can provide valid credentials as rightful owner or users.

3 Discovery and QoS

There is a critical requirement for the clients/devices to be able to discover available Gateway Surrogate Hosts. Absence of a discovery mechanism has the potential to pose as a single point of failure. For reasons of efficiency and fault tolerance, multiple discovery techniques are provided in the architecture. The foremost method of discovery is multicast announcements from Gateway Surrogate Hosts. This automatically provides for locating 'nearby' hosts to the devices (as multicast is often geographically limited to a network boundary by most administrators). HTTP based discovery is provided as a supplement. All available Gateway Surrogate Hosts register with a web service hosted on a known location. Client devices/applications can inquire about a particular host by submitting appropriate parameters to this service over HTTP.

The surrogate paradigm will function most efficiently when the network delays between the device/client and surrogate are minimal. Moreover, efficiency also depends on the proximity of surrogate to the service being accessed. Since the user may be mobile with respect to the Gateway Surrogate Host and Grid resources, support is needed in the architecture to optimize the proximity based parameters. Each Gateway Surrogate Host will keep track of its access quality towards known/available Grid service hosts/networks. On the other hand, before deploying a surrogate, client side application can determine its network connectivity and temporal efficiency with a specific host. This procedure poses a certain one time per start-up burden, but offers better QoS relative to a scenario where such optimizations are left to good luck.

4 Implementation Overview

The authors have provided a bare-bones implementation of the proposed architecture. Before this design is tested for actual Grid service interaction, it is necessary to validate its viability in a general scenario which involves considerable CPU, memory and network utilization. Simple Network Management Protocol [15] is a widely accepted and utilized way of monitoring network entities and we have chosen to verify our approach by monitoring a remote server for 14 system statistics periodically, through a handheld device. Handheld device has network connectivity through a wireless LAN interface. A desktop machine is configured to act as a Gateway Surrogate Host. A Surrogate has been coded for the handheld device with the functionality of monitoring the remote server through SNMP queries and adjusting the results to be sent back to the Device. The results of these queries are to be displayed at the handheld device in the form of dynamic line, bar and pie charts/graphs. Performance of the device and the impact of our executing system will be measured and the benefits and shortcomings of the approach will be highlighted.

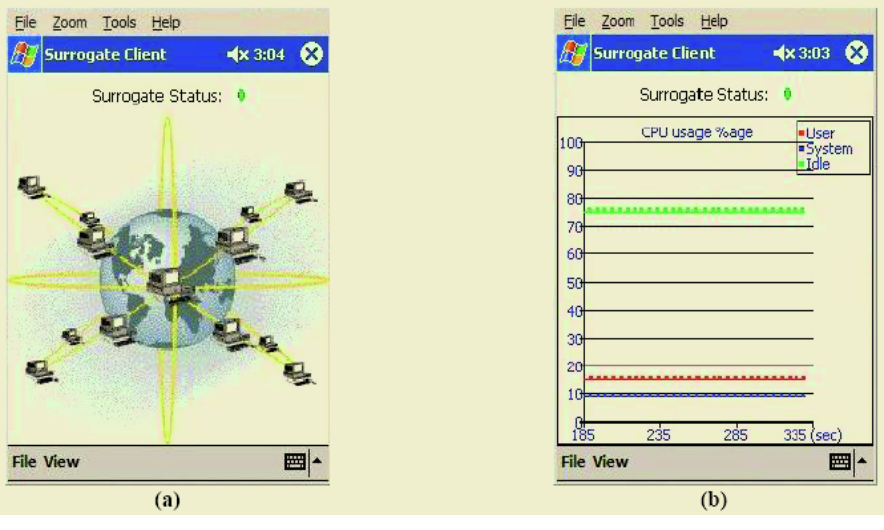


Fig. 2. (a) Main window of the client application executing on the device (b) Remote host's CPU usage statistics by category (user, system, idle) over a period of time, as seen on the device

The Gateway Surrogate Host module has been implemented by modifying and extending the Surrogate Host provided with the reference implementation of Jini Surrogate Architecture specification. The extensions include addition of useful attributes to be announced, additional discovery mechanism and addition of an SNMP agent. IBM's J9 VM for java is used to implement the surrogate

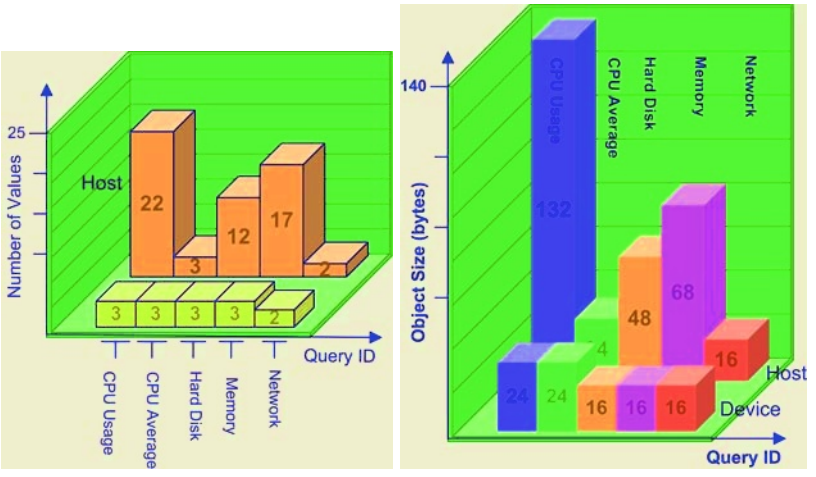


Fig. 3. (Left) Comparison between number of values at the Host and values sent to the Device; (Right) Comparison between size of intermediate results at the Host and size of results at Device

for the handheld device and contains classes which implement the task that the Device wishes to execute. Moreover, it contains the 'device-to-surrogate' interconnect implementation which, in the case of this scenario, is based on IP Interconnect Specification.

4.1 Performance Measurements

Measurements taken to analyze the performance of the Device during the course of execution are presented in Table 1.

Table 1. Result parameter count and size comparison at Gateway Surrogate Host and Device

Query Type	Number of values received at Host	Intermediate result size at Host (bytes)	Number of values sent to Device	Result size sent to Device (bytes)
CPU Usage	22	132+	3	24+2
CPU Avg. Load	3	24+	3	24+2
HDD Utilization	12	48+	3	16+2
RAM Utilization	17	68+	3	16+2
Network I/O	2	16	2	16+2
Total	56	288+	14	106

The size of result object depends on the type of values stored in the fields. The 14 statistical values are received in 5 'Result' objects and amount to, on

average, 62 bytes of results per 5 seconds with additional 44 bytes after every minute. An interesting comparison is made by considering the number of result parameters and their size as retrieved by the surrogate (executing at the Gateway Surrogate Host) with the corresponding values at the Device. A significant amount of information can be condensed by applying intermediate calculations and filtration of values at the surrogate module.

It can be observed that the number of parameters is reduced by 75 percent (4 times reduction) when transferring results to the Device. Similarly, more than 64 percent of the data has been filtered out in intermediate calculations and trimming at the surrogate. This performance markup is in addition to the communication reduction achieved by careful selection of host machine and resources access mechanisms throughout the surrogate's lifetime, as explained earlier. The burden on PDA has been reduced to a few hundred bytes of data and graph formation.

5 Conclusion and Related Work

Research and development for facilitating handheld held devices to interact with Grid services is in early stages. Signal [16] proposes a mobile proxy-based architecture that can execute jobs submitted to mobile devices, in-effect making a grid of mobile devices, but this approach may affect the fault tolerance of the system as the mobile device hosting the proxy also has to deal with the adverse conditions of a mobile/wireless environment. Moreover, the proxy has to schedule the jobs submitted to it by other mobile devices. In our case, the middleware has far more resources at its disposal, so the scheduling can be more flexible and concurrent. GridBlocks [17] builds a Grid application framework with standardized interfaces facilitating the creation of end user services. They argue that SOAP usage at mobile devices may be 2-3 times slower than a proprietary communication protocol, but the advantages of using SOAP (such as overcoming device heterogeneity) may be far more profitable than its limitation.

A solution based on Jini Surrogate Architecture to access Grid services is demonstrated in this paper. In the proposed approach, a resource constraint device wishing to access a resource-demanding service is allowed to delegate this task to a relatively powerful machine (desktop, server). Specifically, CPU intensive, network oriented tasks can be efficiently delegated to such systems when network connectivity is available. In case of intermittent connectivity, applications and services requiring on-demand or periodic network access can benefit from this approach. The implementation has been tested for a moderately intensive task. We intend to extend and implement the architecture to interact with existing Grid services and analyze the performance of our framework. These include HTTP discovery, client authentication, and surrogate migration support. A notable constraints suffered by our approach include the requirement of Java virtual machine at the device. Furthermore, at present we have not addressed the notions of client/surrogate authentication and authorization and are the focus of our future work.

References

1. Foster, I.: What is the Grid? A Three Point Checklist. In: GRIDToday (2002)
2. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. In: Morgan Kaufmann Publishers, San Fransisco (1999)
3. Bunn J., Newman H.: Data-intensive Grids for High-Energy Physics. In: Berman, F., Fox, G.E., Hey, A.J.C. (eds.): Grid Computing: Making the Global Infrastructure a Reality, Wiley (2002). 859-906
4. Hastings, S., Kurc, T., Langella, S., Catalyurek, U., Pan, T., Saltz, J.: Image Processing for the Grid: A Toolkit for Building Grid-enabled Image Processing Applications. In: 3rd International Symposium on Cluster Computing and the Grid, May 12 - 15, 2003, Tokyo, Japan.
5. Breton, V.: Health Grid. In: International Symposium on Grid Computing 2003, Academia Sinica, Taipei, Taiwan (2003)
6. Satyanarayanan, M.: Fundamental Challenges in Mobile Computing. In: Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, Philadelphia (1996)
7. Amendolia, S.R., Brady, M., McClatchey, R., Mulet-Parada, M., Odeh, M., Solomonides, T.: MammoGrid: Large-Scale Distributed Mammogram Analysis. In: Proceedings of the XV111th Medical Informatics Europe conference (MIE'2003). St Malo, France May 2003. Volume 95 of Studies in Health Technology and Informatics, pp 194-199 IOS Press, Amsterdam.
8. Sun Microsystems, Inc.: JiniTM Architecture specification. http://www.sun.com/jini/specs/
9. Sun Microsystems, Inc.: JiniTM Technology Surrogate Architecture Specification. http://surrogate.jini.org/sa.pdf (2003)
10. S. Vazhkudai, S., Tuecke, S., Foster, I.,:Replica Selection in the Globus Data Grid. In: Proceedings of the first IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001), IEEE Computer Society Press,(2001) 106-113,
11. Lee, B., Weissman, J.B.: Dynamic Replica Management in the Service Grid. In: High Performance Distributed Computing 2001 (HPDC-10"01), San Francisco, California (2001) p. 0433
12. Sun Microsystems, Inc.: JiniTM Technology IP Interconnect Specification. http://ipsurrogate.jini.org (2001)
13. Lee, S., Gerla, M.: Dynamic Load-Aware Routing in Ad hoc Networks. In: Proceedings of The Third IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET 2000), Richardson Texas (2000)
14. Godfrey, B., et al.: Load Balancing in Dynamic Structured P2P Systems. In: IEEE INFOCOM 2004, Addis Ababa, Ethiopia (2004)
15. Stallings W.: SNMP, SNMPv2, SNMPv3, and RMON1 and RMON2. 3rd Edition Addison-Wesley, California (1999) 71-82
16. Hwang, P. Aravamudham Middleware Services for P2P Computing in Wireless Grid Networks. In: IEEE Internet Computing vol. 8, no. 4, July/August 2004, pp. 40-46
17. Gridblocks project (CERN) http://gridblocks.sourceforge.net/docs.htm

An Extendable GRID Application Portal

Jonas Lindemann and Göran Sandberg

LUNARC, Center for Technical and Scientific Computing, Lund University
jonas.lindemann@byggmek.lth.se
http://www.lunarc.lu.se

Abstract. To attract users from a wide range of scientific areas it is important to provide a variety of ways to access the Lunarc resources. Providing good user interfaces for all categories of users is a key factor for a high use of resources. One way of doing this is by providing access to common applications through a web-based portal. Existing grid portals are often geared toward's providing access to the features of the middleware instead of providing interfaces to existing applications. Lunarc has adopted a more application oriented approach providing dedicated user interfaces for each application using the grid middleware as an infrastructure for the implementation. The users as application experts can extend the portal themselves using an easy to use software development kit (SDK) for plugins.

1 Introduction

As more research areas need more computational resources, it is very important to provide easy access to these resources, both via web-based interfaces and from the command-line. Currently, most users of the Lunarc resources use command-line tools for accomplishing tasks. For other scientific areas this way of accessing a system can be very unfamiliar. To complement the existing command-line-based user interfaces for the Lunarc resources it was decided to implement a web-based portal to provide easy access to commonly used applications.

At first it was thought that existing grid portals could be used. Unfortunately, most existing grid portals have a focus on providing access to the specific features of the middleware instead of the applications. To change the focus back to the users and application, the Lunarc Application Portal has been developed. In this portal the application is the focus of the user interface and the grid middleware is the infrastructure for achieving this goal.

There exists today a number of frameworks for implementing grid-portals, gridsphere[1], CrossGrids[2], GridBlocks[3] and many more. These are often designed in a very general way providing many rich features to the grid-middleware. The underlying applications are often specified as an option on the job submission form. The user is also often required to have technical knowledge of many grid features to submit a job to the grid. To remedy this one can argue that these portals can be extended. This is true, but adding functionality to many of these portals often require deep knowledge of object-oriented concepts and web

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1012–1021, 2005.

development, which limits the development of application specific extensions to experts in these areas, adding an extra layer between the application developer and the portal end user. At Lunarc, a different approach has been taken. Instead of focusing on all the possibilities in the grid-middleware the focus has been on how to make an application available to the end-user in an easy and effective way, so that the user can take advantage of the resources available on the grid.

To make a web portal useful the users must be involved in the process of creating the interfaces used. Many of our users are application experts in that they often develop their own simulation software. To make these users involved in the development of the application portal the portal also provides an easy to use Python-based software development kit. With this kit users can develop plugins in an easy way without extensive knowledge in object-oriented programming or libraries.

2 User Interface

A web-based user interface was chosen, as it can be accessed from most platforms. It also eliminates the need for any installations on the client side. Web applications are also a familiar concept that most users are able to handle.

The user interface for the Lunarc application portal is shown in figure 1, The interface is divided in two parts, a menu bar and a work area. Forms and status messages are shown in the workarea when the user selects functions from the menu. To support the user, the main categories in the menu are ordered so that it reflects the users normal workflow. Drop down menus are used to give the user interface the look and feel of an normal window-based application.

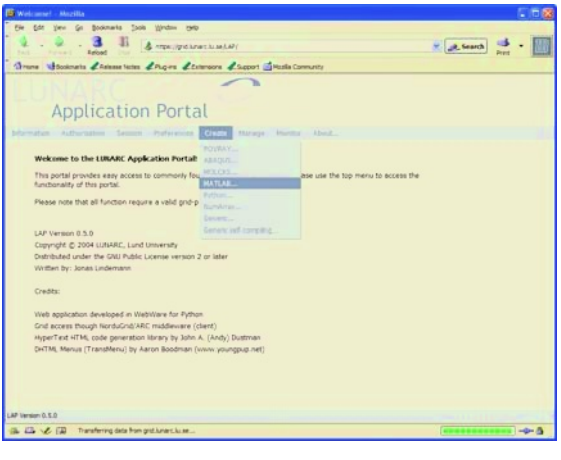


Fig. 1. Lunarc Application Portal

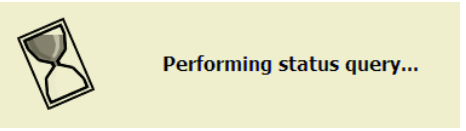


Fig. 2. Progress messages

Effort has been taken to provide rich feedback whenever possible, for example providing status messages for operations that take a long time to complete, see figure 2.

There are 7 main menus: Information, Authorisation, Session, Preferences, Create, Manage and Monitor. The "Information"-menu contains information on how to get started with the portal, a user's guide and a guide for writing plug-ins for the portal. The "Authorisation"-menu contains functions requesting certificates and requesting access to the Lunarc grid systems. The "Preferences"-contains special user preferences, such as default email address, preferred clusters etc. The most important menus are the "Create", "Manage" and "Monitor". From the "Create" menu different job types can be created. Each menu is provided by a special application plugin responsible for providing the application with a user interface and the procedure for submitting the job on the grid. The "Manage"-menu handles already created jobs and jobs running on the grid resources. The "Monitor"-menu contains functions for monitoring running jobs and the monitor the status of the Lunarc clusters.

The main activities in the portal are creating job definitions, submitting jobs, monitoriing and retrieving results from the jobs as shown in figure 3.

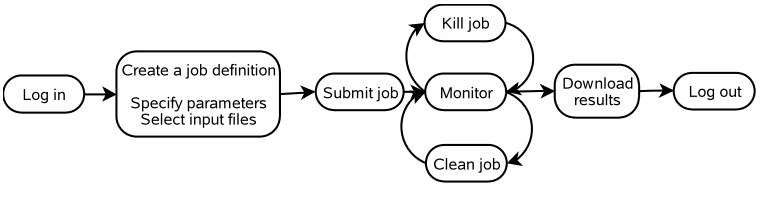


Fig. 3. Portal workflow

2.1 Job Definitions

Before a job can be submitted to the grid a job definition must be created. A job definition is a directory on the portal server containing the needed input files and the settings for the job. The job definition also contains results retrieved from the job after successful completion. Job definitions are created from the "Create"-menu. When the user selects a job definition type a form is shown with the needed attributes for the application type. Figure 4 shows the needed attributes for a simple ABAQUS simulation. Job definitions are managed from

the “Manage”-menu. From this menu job definitions can be submitted to the grid, modified and results can be viewed, see figure 5.

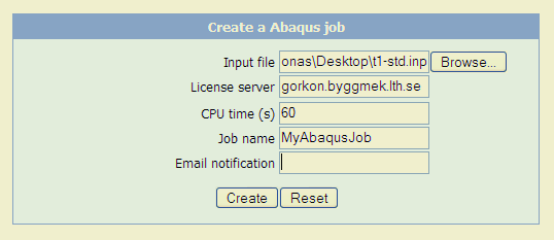


Fig. 4. ABAQUS application attributes

Fig. 5. Manage job definitions

2.2 Managing Running Jobs

Jobs that have been submitted to grid resources are also managed from the “Manage”-menu. When selecting “Manage/Running jobs...” a list of all jobs submitted to the grid are shown along with the operations that can be performed on them, such as getting results (Get), killing a running job and cleaning a deleted job, see also figure 6.

Fig. 6. Manage running jobs

2.3 File Handling

File handling is an important part of an application portal as most application often generate a large amount of output files. The Lunarc Application portal implements a simple web-based file manager to handle viewing and downloading of generated result files. The user interface of the file manager is similar to the those found in normal window-based user interfaces. As seen in figure 7 there are functions for downloading individual files and compressing the entire folder and downloading it as a `.tar.gz`-file.

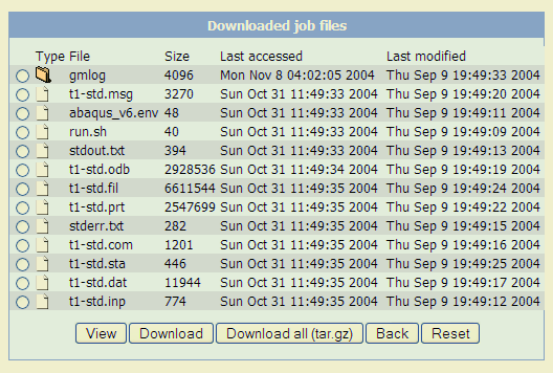


Fig. 7. Documentation provided by the portal

2.4 Documentation

To ease the learning curve, the portal provides it own documentation form the "Information"-menu, see figure 8. 3 main documents are provided: A getting started manual, a user's guide and a programmer's guide. The getting started manual describes briefly how to get started with the portal. The user's guide provides more detailed information on all the functions of the portal. The programmer's guide provides information on how to write plugins for extending the portal with additional application user interfaces. The "Information"-menu also provides information about any extra software needed, such as compression and proxy tools.

Fig. 8. Documentation provided by the portal

3 Implementation

The technical implementation aspects of the Lunarc Application portal is focused on creating a lightweight and easily extendable portal. An important aspect of this is not to reinvent the wheel. Existing software packages and tools are used as much as possible. The Lunarc Application Portal is implemented in Python [4] using the WebWare application server [6]. The application server is hosted in the Apache web server [5]. Job submission and management is done using the NorduGrid/ARC grid middleware [7]. The architecture is illustrated in Figure 9.

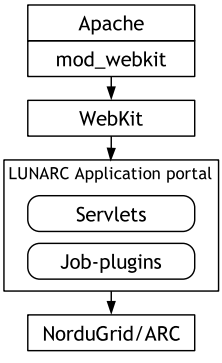


Fig. 9. Application architecture

The WebWare is an object-oriented multithreaded application server capable of handling multiple user web sessions. Each page on the server is represented by a Python class. The application server can be integrated into the Apache Web server by using a cgi-bin version or using the faster mod_webkit apache module. This also the method used in the Lunarc Application portal.

3.1 Grid Middleware

The application portal uses the ARC middleware for job submission, management and monitoring. The portal currently interfaces with the grid middleware by calling the existing command line tools provided by ARC, as there are no python bindings. This approach can be complicated and fragile as the output from the command line tools needs to be parsed. Later versions of the portal will use ARCLib, a library with SWIG generated bindings for many script languages. Using this library will enhance the stability and reduce the complexity of interfacing with the ARC middleware. The ARCLib will be released with future ARC releases.

Currently the interface to ARC is implemented in the python module pyARC using the DN, Proxy and Ui classes. The DN class is just a simple class for parsing

a Distinguished Name (DN). The Proxy class implements a class for handling a proxy. The main ARC interface is implemented in the Ui class.

3.2 Web Implementation

There are two major base classes in the Lunarc Application portal. The LunarcPage class defines the basic layout of all other pages in the portal. All pages requiring a login are derived from the SecurePage class. This class is in turn derived from the LunarcPage. SecurePage derived pages will check that the user has provided a valid proxy, if not the user is provided with a dialog to select it. When this is done the actual page is shown.

In the session directory user preferences are stored in files using the pickle() and unpickle() functions. Created jobs and uploaded files are stored in special job directories with the `job_xxxx` where `xxxx` indicates the name of the job.

3.3 Adding Functionality Using Plugins

One of the reasons for choosing Python as the implementation language is to make it easier for user groups to create their own custom application interfaces. Extending the portal in this way requires the plugin-author to provide two classes. One class derived from the JobPage-class implementing the needed user interface for the plugin. A second class derived from the Task-class implementing the needed procedures for setting up a job for submission to grid resources, see figure 10.

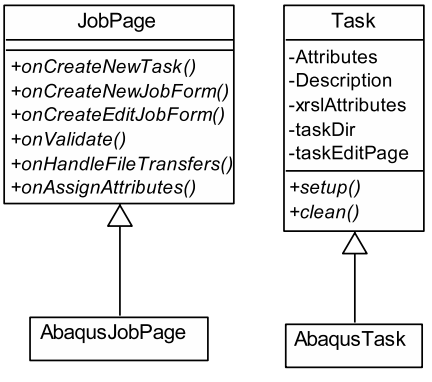


Fig. 10. Implementing extension classes

The following code excerpt shows how the setup-method for the AbaqusTask class is implemented:

```
def setup(self):

    # Get directory and attributes
    taskDir = self.getDir()
    attribs = self.getAttributes()

    self.addInputFile(attribs["inputFile"])

    # Create abaqus env file
    envFile = file(os.path.join(taskDir,"abaqus_v6.env"), "w")
    envFile.write(abaqusEnvFileTemplate %
        {"licenseServer":attribs["licenseServer"]})
    envFile.close()

    # Create shell script
    jobIdentifier, ext = os.path.splitext(attribs["inputFile"])

    shellFile = file(taskDir+"/run.sh", "w")
    shellFile.write(abaqusShellTemplate %
           {"jobIdentifier":jobIdentifier})
    shellFile.close()

    # Create XRSL file

    xrslFile = LapJob.XRSLFile(self)
    xrslFile.setFilename(taskDir+"/job.xrsl")
    xrslFile.write()
```

The JobPage is responsible for defining the needed user interface for the application. This is done using the python module LapWeb, which contains classes for defining user interface components. An excerpt from the the onCreateNewJobForm is shown below:

```
def
    onCreateNewJobForm(self, task):

    form = LapWeb.Form(self, "testform", "AbaqusJobPage",
        "Create a Abaqus job")

    attribs = task.getAttributes()
    xrslAttribs = task.getXRSLAttributes()

    form.addFile("Input file", "inputFile", "")
    form.addText("License server", "licenseServer",
        attribs["licenseServer"])
    form.addSeparator()
    form.addText("CPU time (s)", "cpuTime", xrslAttribs["cpuTime"])
    form.addText("Job name", "jobName", xrslAttribs["jobName"])
    form.addText("Email notification", "email", xrslAttribs["notify"])

    return form
```

4 Security

One of the more complicated procedures when using grid-based resources is security. Before the event of the grid, to use a cluster all that was needed was a user name and a password. This model works well on a single resource, but in a larger context, problems with password security becomes apparent. To remedy this, grid middleware uses a different security model based on private keys and certificates. To access grid resources the user must have a signed certificate. In addition to this, his grid ID must be added to the sites that he has been authorised to run at.

4.1 Certificate and Proxy Handling

The solution adopted by the Lunarc Application Portal is described below:

1. The user applies for an account on the Lunarc systems.
2. Generate private key and certificate request using the ARC GRID middleware (grid-cert-request). This is preferable done on the users own system.
3. The certificate request is sent to the NorduGrid CA for signing.
4. The signed certificate is installed on the client system.

The above solution requires the ARC GRID middleware to be installed on the client system. If the client system is not supported, a special version of the grid-cert-request tool has been developed to make it possible for users to generate a request and a private to be able to log in to the portal.

Access to the portal is done through a GRID proxy. This proxy is generated using the grid-proxy-init command line tool or using the special GUI proxy generation tool from the Java CoG 1.1 toolkit [8]. Future version of the portal will probarbly use a MyProxy[9] server for authentication instead of using a proxy-based login.

4.2 Access to Resources

Access to Lunarc grid resources goes through a special application from where the user requests access. The user also has to supply his existing Lunarc account, so that his usage will be logged correctly. When all information has been checked, the user is added to the Lunarc VO and the NorduGrid VO.

5 Conclusions

Not all scientific groups share the same computing culture. To some groups command line tools and classical HPC usage is very unfamiliar. A web-based portal can complement the existing command line user interface to give more groups access to computational resources.

By focusing on application functionality instead of middleware functionality, users will recognise the application interfaces instead of having to learn the

details of job submission before being able to submit a job. In this model grid middleware is the infrastructure for achieving this goal.

The success of a web portal depends greatly on the acceptance of the users. This requires that the architecture of the portal is flexible and adaptable to user needs. This is achieved by using a Python-based lightweight application server reusing existing component and tools such as the ARC middleware and by providing users means of extending the portal themselves using a plugin-based approach.

References

1. gridphere portal framework, http://www.gridsphere.org/gridsphere/gridsphere, 2005
2. Developing new Grid components, http://www.crossgrid.org/main.html, 2005
3. GridBlocks, http://gridblocks.sourceforge.net, 2005
4. Python, http://www.python.org, (2004)
5. The Apache HTTP Server Project, http://httpd.apache.org, (2004)
6. Python Web Application Toolkit, http://www.webwareforpython.org, (2004)
7. Advanced Resource Connector ARC, http://www.nordugrid.org, (2004)
8. Java CoG Toolkit, http://www.cogkit.org, (2004)
9. MyProxy Online Credential Repository, http://grid.ncsa.uiuc.edu/myproxy, 2005

A Task Replication and Fair Resource Management Scheme for Fault Tolerant Grids

Antonios Litke, Konstantinos Tserpes, Konstantinos Dolkas, and Theodora Varvarigou

Department of Electrical and Computer Engineering,
National Technical University of Athens,
9, Heroon Polytechniou Str, 15773 Athens, Greece
{ali, tserpes, dolkas, dora}@telecom.ntua.gr
http://telecom.ece.ntua.gr/

Abstract. In this paper we study a fault tolerant model for Grid environments based on the task replication concept. The basic idea is to produce and submit to the Grid multiple replicas of a given task, given the fact that the failure probability for each one of them is known a priori. We introduce a scheme for the calculation of the number of replicas for the case of having diverse failure probabilities of each task replica and propose an efficient resource management scheme, based on fair share technique, which handles the task replicas so as to maintain in a fair way the fault tolerance in the Grid. Our study concludes with the presentation of the simulation results which validate the proposed scheme.

1 Introduction

Grid can be an appropriate solution for many computational intensive and grand scale applications ranging from scientific, industrial and engineering field. It is also an emerging solution for utility and pervasive computing. However, Grids as all the large scale distributed platforms are prone to failures, which restrain it to become a reliable execution platform for high performance and distributed applications. So the fault tolerance feature is of vital importance. By the term *fault tolerance* we denote the ability of the Grid system to perform correctly in the presence of faults.

Fault tolerance is of big importance in Grid computing, as the emerging grid-oriented applications have a significantly increased size and complexity from the traditional ones. Experience has shown that systems with interacting and complex activities are inclined to errors and failures. Thus, Grid computing is not expected to be fault free, despite the fact that individual techniques such as *fault avoidance* and *fault removal* [1] may additionally be applied to its resources. The fault tolerance feature is introduced in the Grid systems in order to enhance them with the appropriate reliability, which is mandatory in the context of diverse, dependable and cross-organizational environments. The reliability in Grid comprises the probability of all grid applications to be executed fully with no errors in the grid computing environment. As applications scale to take advantage of a Grid's vast available resources, the probability of failure is no longer negligible and must be taken into account.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1022 – 1031, 2005.

There are various approaches to make grid computing fault tolerant [1],[2],[3]. The basic however are the checkpoint recovery and the task replication. The former is a common method of ensuring the progress of a long-running application by taking a checkpoint, i.e., saving its state on stable storage periodically. A checkpoint recovery is an insurance policy against failures. In the event of a failure, the application can be rolled back and restarted from its last checkpoint—thereby bounding the amount of lost work to be recomputed. Task replication is another common method that aims to provide fault tolerance in distributed environments by scheduling redundant copies of the tasks, so that to increase the probability of having at least a simple task executed. A brief overview of the options in the fault tolerant computing on the Grid can be found in [2].

There has been a variety of implementations that have addressed the problem of fault tolerance in Grid and distributed systems. Globus [15] provides a heartbeat service to monitor running processes to detect faults. The application is notified of the failure and expected to take appropriate recovery action. Legion [16] provides mechanisms to support fault tolerance such as checkpointing. Other Grid systems like Netsolve [17], Mentat [18] and Condor-G [19] have their failure detection mechanisms and their failure recovery mechanisms. They provide a single user-transparent failure recovery mechanism (e.g. retrying in Netsolve and in Condor-G, replication in Mentat). The difference between these systems and our proposed scheme relies on the fact that the one presented here addresses the fault tolerance as a metric that is adjusted in a *fair* way for all the Grid users. Moreover it applies to all Grid environments and is especially beneficial for low *workload* jobs in unreliable environments, such as Mobile Grids [20], which consist of mobile resources (hosts and users) connected by wireless links and forming arbitrary and unpredictable topologies.

In this paper we study a fault tolerant model for Grid environments based on the task replication concept. The basic idea is to produce and schedule in the Grid infrastructure multiple versions (replicas) of a given task, based on the fact that the failure probability for each one of them is known a priori. The replication model that is adopted is based in static replication [4] meaning that when a replica fails it is not substituted by a new one. The failure of a task replica is based on aspects that concern the task itself and not the resource on which it is going to be executed. This approach implies that the Grid infrastructure remains unchanged concerning its topology and total computational capacity, and independent from the faults that occur in the Grid environment. The introduction of task replicas causes an overhead in the workload that is allocated for execution to the Grid environment. Moreover, scheduling and resource management are important in optimizing Grid resource allocation, and determining its ability to deliver the negotiated QoS and provide fair access to all users [5][3]. The basic idea that is applied in this study is to address the additional overhead caused by the task replicas in the Grid system with a fair scheme of resource management that will provide a fair share of computational resources to the Grid users [8][9].

The remainder of this paper is structured as follows: Section 2 provides the problem formulation for the fault tolerance and task replication in the Grid and the notation that will be used. Section 3 provides the task replication model for tasks' whose failure probability is a random function. In section 4 we describe the need for adopting a mechanism for the efficient handling of the additional load that has been caused

by the replicas and which is based on the *max min fair share* scheme, aiming to satisfy as many as possible users with the available resources. Finally, in section 5 we present the simulation results of the developed scheme and conclude, in section 6, with a discussion on future work as well as on potential improvements and enhancements on our proposed scheme.

2 Notation and Problem Formulation

We consider that a set of M processors forms a Grid infrastructure. Each processor has a fixed computational capacity denoted as $c_j, j \in \{1,2,...,M\}$, thus the total computational capacity of the Grid is $C = \sum_{j=1}^{M} c_j$. We also consider a set of N different tasks $T_i, i \in \{1,2,...,N\}$ to be assigned to the Grid for execution. We assume that the tasks are non-preemptable and non-interruptible [10]. This means that a task cannot be broken into smaller sub-tasks or modules and it has to be executed as a whole on a single processor. Additionally as soon as a task starts its execution on a processor, it cannot be interrupted and it consumes the whole processor computational capacity as long as it is executed.

Each task T_i has an *execution time* ET_i and a *deadline* D_i. The execution time corresponds to the time interval that the execution of T_i lasts if it is executed in a processor of unitary *capacity* $c = 1$. The deadline of the task represents the latest time at which the Grid has to deliver the results to the user. It is a quantity specified by the end-user who is willing to pay for the Grid resources used. During a task execution on the Grid, various errors might occur causing task failure. In this study we will deal with those cases that are based upon the distributed systems fault model, which includes omission, timing and arbitrary faults [12]. These kinds of errors are commonly met in distributed systems as well as in Grid environments. We will omit other types of failures such as hardware failures [2][4][13], etc.

We define the *failure probability* Pf_i of a task T_i, which is the probability that the task fails to be executed on the Grid. Respectively, *success probability* Ps_i is the probability that the task T_i concludes its execution within the Grid system, providing the presumable results. The correlation between failure probability and success probability is:

$$Pf_i = 1 - Ps_i . \tag{1}$$

At this point we introduce the concept of *workload*. Workload $w_i, i \in \{1,2,\text{K},N\}$ is the computational capacity that is required by a task T_i in order to be executed in unitary time on a given resource. For simplicity reasons we have focused only on computational capacity and we have omitted other parameters such as communication delays, disk input/output delays etc. Moreover, in our study we have reduced the computational capacity into unitary, when referring to the workload, in order to treat the resources as homogeneous simplifying thus the presented model. However, the

extended scheme incorporating various heterogeneous resources can be derived in the same manner. The workload is equal to the inversed execution time ET_i, and can be written as:

$$w_i = (ET_i)^{-1} . \tag{2}$$

Task replicas are generated and assigned to the Grid for execution. The term *replica* or *task replica* is used to denote an identical copy of the original task. By producing task replicas, a low probability of task failure can be achieved. We assume that m_i replicas –denoted by $T_{ik}, k = 1,\text{K}, m_i$- of a task T_i are produced and are placed among other tasks and replicas that are to be submitted for execution. Given the failure probability Pf_{ik} of each one of the m_i replicas T_{ik} of task T_i, the failure probability is defined as:

$$Pf_i = \prod_{k=1}^{m_i} Pf_{ik} . \tag{3a}$$

The above corresponds to the probability of the event "all task replicas fail". Respectively, the success probability Ps_i is equal with the probability of the event "at least one task replica succeeds" and is given by the equation:

$$Ps_i = 1 - Pf_i = 1 - \prod_{k=1}^{m_i} Pf_{ik} . \tag{3b}$$

It can be assumed that the similarity of the replicas implies that the variations of the failure probabilities of each one of them cannot be large. Given that assumption, we can define two bounds for the failure probability Pf_{ik} of each replica T_{ik}, namely u_i and l_i are the maximum and the minimum value that Pf_{ik} can take for each of the replicas of T_i. These two bounds can be either estimated by the Grid user or can be statistically determined by the previous history of the system according to the relative tasks that have been already submitted. Alternatively, prediction models can be applied for the estimation of the failure probabilities based on the individual task features in a similar way as described in [14].

In order to guarantee a low failure probability our scheme produces as many task replicas as needed so as to satisfy the constraint of success probability. We now define a *probability threshold* δ, which denotes the probability that each task (including its replicas) will not finish its execution. We can write:

$$Pf_i \le \delta . \tag{4}$$

where δ is a constant between 0 and 1.

3 Task Replication Model

We will present a way to calculate the number of replicas that is required in order to secure a fault tolerant operation of the Grid for all cases. We will distinguish between

two cases that will be examined in this section. In the first case it is assumed that two different positive numbers u_i, l_i bound the failure probability Pf_{ik} of a replica T_{ik}. In the second case Pf_{ik} is unbounded and it can take random values, so a simple algorithm is used to produce replicas in order to follow inequality (4).

First case: $l_i < Pf_i < u_i$

The failure probability Pf_{ik} for each replica of a task can be bounded by two positive real numbers u_i, l_i. From (3a) and (4) we have:

$$l_i^{m_i} \le u_i^{m_i} \le \delta, \tag{5}$$

and in the sequel:

$$m_i \le \frac{\log(\delta)}{\log(u_i)} \Rightarrow m_i \le \left\lfloor \frac{\log(\delta)}{\log(u_i)} \right\rfloor. \tag{6}$$

In the simple case of a constant failure probability $Pf_i = F_i = const$, it becomes:

$$m_i \le \left\lfloor \frac{\log(\delta)}{\log(F_i)} \right\rfloor. \tag{7}$$

Second Case: Pf_{ik} is a function of *k*.

We use a simple algorithm to specify the number of the needed replicas for each task. The idea is to produce a replica of a task each time the failure probability is bigger than δ. The replication procedure stops when inequality (4) holds true for a given number of replicas m_i. The proposed algorithm computes the number of replicas to be produced for a task T_i in order to reduce the failure probability Pf_i at least bellow the value δ.

The replication procedure for both cases takes place in the *Grid middleware*, which is responsible for keeping the level of fault tolerance for the whole Grid environment. This approach does not make necessary any communication between the replicated tasks themselves. The presented scheme, however, does not exclude any parallel programming applications from being executed on the specific platforms, since it does not comprise a constraint to these tasks. Moreover, although the failure probability is attached to tasks, this assumption does not affect the generality of our approach, since in the second case, where the failure probability is a random function of *k,* we can assign to *k* values that are dependent to the resource itself.

4 Efficient Fault Tolerant Mechanism with Fair Share

We assume that the workload of each task is the sum of the workloads of its replicas, which can be written as $w_i = \sum_{k=1}^{m_i} w_{ik}$ or $w_i = m_i \cdot w_{i1}$, since every replica is identical

to the primary task which was assigned by the user. In this way every task can be considered as a *virtual task*, comprising as the set of itself and all of its replicas and which has a workload equal to an integral multiple of the original's workload.

We propose a *max min fair share* mechanism for the management of the fault tolerant feature of each task, by reducing in a fair way the replicas of each task. It is important to clarify the difference between the demanded capacity d_i and the capacity provided by the Grid which will be symbolized with a_i. In our case we will consider as required capacity the workload w_i. The number of replicas actually allowed by the Grid is other than the one required (m_i) to guarantee the fault tolerance of the Grid. We denote the number finally assigned replicas as n_i. This fact raises the need to determine the new number of replicas that actually can be assigned in order to have the maximum possible probability threshold δ which utilizes the total of the Grid capacity so as to satisfy the tasks' deadlines. We will apply a *max min fair share* technique [8][9] to determine the fair share that can be allocated at each task.

The main idea of this scheme is that all users submitting their jobs in the Grid are allocated an equal share of the computational resources for the execution. Although many schemes can be adopted for the efficient handling of Grid resources, we deal in this study with the fair share model which is a decent way to share resources without having prioritized and weighted clients. In case where a specific job does not require all its assigned computational power, the remaining part is being distributed evenly to the remaining users in a recursive way. In order to classify the tasks that are submitted in the Grid according to the required computational power, we need to define the quantity of demanded capacity d_i which, in our case, corresponds to the workload w_i. We will refer to that amount as *fair share x*.

The demanded capacity for every task is compared with the fair share x. If the task's computational capacity allocated is bigger than or equal to the fair share, then it is considered as a *satisfied* task. In the other case, the task is classified as *unsatisfied* (the capacity given is smaller than the fair share). In the sequel the fair share is being recalculated. The new fair share accrues from the capacity U, which is remaining to be shared among the unsatisfied tasks. This algorithm runs iteratively until either all tasks are satisfied with the demanded capacity, or there is no remaining capacity U to be distributed. The proposed scheme assigns in a fair way the total capacity of the Grid to the virtual tasks that are submitted for execution. The virtual tasks that have received no sufficient resources for their execution can either be scheduled in a later phase having as disadvantage the deviation from the given deadline D_i.

5 Simulation Results

The proposed efficient fault tolerant mechanism has been implemented in C++ and evaluated against a set of tasks. In the following table we present the tasks that have been used for the evaluation of the proposed scheme. The tasks have been selected so as to provide a main separation between "reliable tasks" with a low failure probability Pf_i between 0 and 0.15 and those that have a higher failure probability between 0.2

and 0.35. The failure probabilities of the individual tasks have been randomly generated between each of the two values respectively. The workload of each original task ranges between 1 and 100 computational units. By this way, a mean value of 50 computational units can be assigned to each one of the original tasks, which in turn leads to a "hard" scenario, given the fact that the produced replicas will augment the demanded capacity overall. As fault tolerance threshold δ we have selected the value of 0.05. This threshold has been selected so as to provide a high degree of reliability, higher than the mean value of the "reliable" tasks.

Table 1. Input provided for the simulation results of the proposed fault tolerant scheme

Fault tolerance threshold δ =0.05			
No of tasks	**No of "reliable" tasks**	**Grid's Capacity *C***	**% of satisfied tasks**
30	20	1500	80%
50	30	2500	52%
100	50	5000	63%
200	120	10000	85%
500	300	25000	65%
1000	400	50000	88.2%

Figure 1 shows the relation of the workload generated for each T_i *before* and *after* the use of the max min algorithm. Figure 2 depicts the relation between the failure probability Pf_i *before* and *after* the use of the max min fair share algorithm for each virtual task. Examining the presented results, we can see that in some cases there is not enough workload assigned to a virtual task, leading to the incapability of the procedure to provide service even to a single replica. This fact is clearly depicted in Fig. 2 where the failure probability after the max min fair share algorithm reaches to 1, which is interpreted as a complete failure to serve the certain task which actually means its rejection from the fault tolerant Grid and, consequently, its deviation from the user specified deadline. This rejection, although undesired, does not imply inefficiency of the proposed scheme, since the basic motivation of the work is to construct a mechanism of providing a fault tolerant Grid for both "reliable" and "less reliable" tasks and not to guarantee the achievement within the given deadline.

The result analysis in Fig. 1 shows that even in the case where the tasks are equally distributed among "reliable" and "less reliable", the tasks that are finally blocked are 3, while less than 10% of the tasks have a failure probability between 0.2 and 0.3. Again, the majority of the tasks, comprising the 80% of the total tasks assigned, is successfully scheduled in the Grid with a failure probability less or equal to 0.05, although their initial failure probability was significantly higher before applying the proposed scheme.

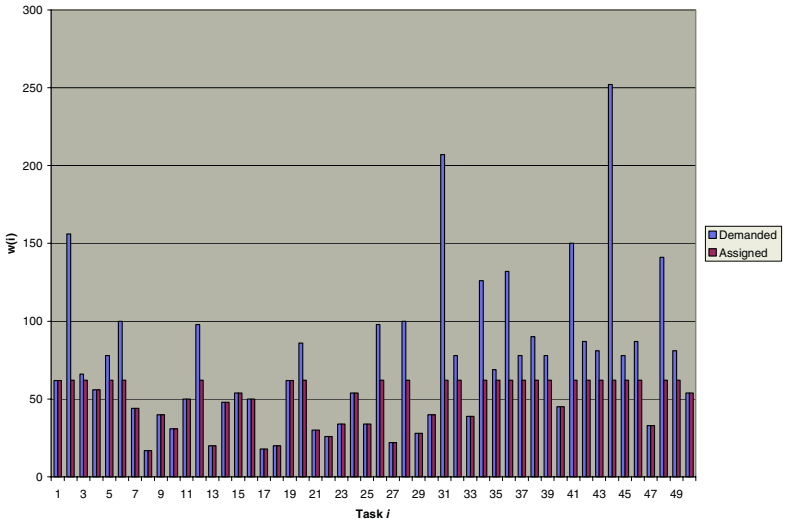


Fig. 1. The case of 50 different tasks with 30 reliable ones and fault tolerance threshold δ = 0.05. The respective workloads of the virtual tasks as resulted for the *demanded* and *assigned* case for δ = 0.05

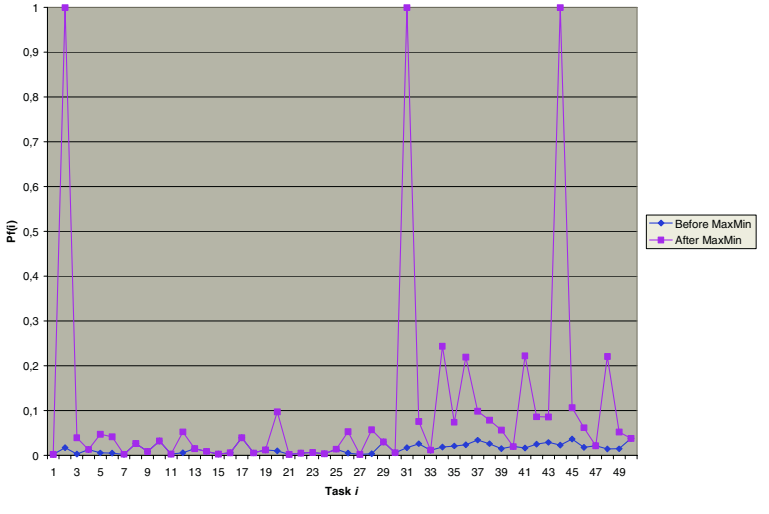


Fig. 2. The case of 50 different tasks with 30 reliable ones and fault tolerance threshold δ = 0.05. The failure probability for each task *before* and *after* the max-min fair share technique

From the workload perspective the maximum deviation between the primary demanded and the eventually assigned workload to each set of tasks ranges between 43.43% of the demanded workload in the set of the 1000 tasks and 77,54% in the set

comprising of 500 tasks. The largest number of not satisfied tasks in terms of workload is presented in the set of 50 tasks. In particular, 24 out of 50 tasks are not provided with the required amount of workload by the proposed mechanism. The average percentage of the deviations in the demanded workload of the 50 tasks set is 17.09%, a rather small number, if we consider the overall required satisfaction level of fault tolerance threshold. Some results of special interest are those of the biggest set of 1000 tasks. In that case, 88.2% of the tasks are satisfied in terms of assigned workload while at the same time the average deviation of the assigned workload is 2.89% of the demanded workload.

6 Conclusions

In this paper we have studied a task replication scheme that applies max-min fair share resource management for providing fault tolerance in Grids. The main contribution of this work relies in performing task replication by using the proposed algorithm which is designed for the case of having diverse failure probabilities between a task and its replicas, and in the handling of the fault tolerance fair share for the efficient assignment of the tasks in the Grid. The scheme has been implemented and validated for a variety of tasks with a diverse set of failure probabilities for the given tasks and their replicas. It showed that in cases where we have tasks equally distributed among "reliable" ones and "less reliable" ones, a fault tolerant Grid can be achieved by having rejected only a small number of tasks resulting in their deviation from their deadlines. The other tasks and their replicas are successfully scheduled in the Grid providing thus a high degree of fault tolerance. The presented results although in a preliminary form, are indicative for the evaluation of the proposed scheme. The approach that is presented can be further improved by taking into consideration the deviation from the deadline for each task and assuming this deviation as a criterion for prioritized scheduling.

References

1. M.R. Lyu,, Software Fault Tolerance, John Wiley & Sons – Chichester, 1995
2. J. B. Weissman. Fault Tolerant Computing on the Grid: What are My Options? HPDC 1999
3. F. Wang, K. Ramamritham, J.A. Stankovic. Determining redundancy levels for fault tolerant real-time systems, IEEE Trans. Computers, vol 44, issue 2, 1995, pp. 292-303
4. A. Nguyen-Tuong. Integrating Fault-Tolerance Techniques in Grid Applications, PhD Dissertation, University of Virginia, August 2000
5. Scheduling Working Group of the Grid Forum, Document: 10.5, September 2001
6. K. Ramamritham, J.A.Stankovic, and P.-F. Shiah. Efficient Scheduling Algorithms for Real-time Multiprocessor Systems, IEEE Trans. on Parallel and Distributed Systems, vol.1, no.2, 1990, pp.184-194
7. L. E. Jackson and G. N. Rouskas. Deterministic Preemptive Scheduling of Real Time Tasks, IEEE Computer, vol. 35, no. 5, 2002, pp. 72-79
8. A. Demers, S. Keshav and S. Shenker, Design and Analysis of a Fair Queuing Algorithm, Proc. of the ACM SIGCOMM, 1989

9. D. Bertsekas, R. Gallager, Data Networks, Prentice Hall, 1992. The section on max-min fairness starts on p.524
10. J.Y-T. Leung and M.L. Merrill, A Note on Preemptive, Scheduling of Periodic, Real-Time Tasks, Information Processing Letters, 11, no. 3, 1980, pp. 115-118
11. M. L. Dertouzos and A.K.-L. Mok, Multiprocessor On-line scheduling for Hard Real Time Tasks, IEEE Trans. on Software Eng., vol. 15, no. 12, 1989, pp. 1497-1506
12. A. S. Tanenbaum, M. van Steen, Distributed Systems: Principles and Paradigms, Prenctice Hall, Computer Science, 2002
13. T. Varvarigou, J. Trotter, Module replication for fault-tolerant real-time distributed systems, IEEE Transactions on Reliability, vol. 47, no. 1, 1998, pp. 8-18
14. N. Doulamis, A. Doulamis, A. Panagakis, K. Dolkas, T. Varvarigou and E. Varvarigos, A Combined Fuzzy -Neural Network Model for Non-Linear Prediction of 3D Rendering Workload in Grid Computing, IEEE Trans. on Systems Man and Cybernetics, Part-B (accepted for publication)
15. The Globus project. http://www-fp.globus.org/hbm/
16. A. Nguyen-Tuong, and A.S. Grimshaw, "Using Reflection to Incorporate Fault-Tolerance Techniques in Distributed Applications," Computer Science Technical Report, University of Virginia, CS 98-34, 1998.
17. H. Casanova, J. Dongarra, C. Johnson and M. Miller, "Application-Specific Tools", in I. Foster and C. Kesselman (eds.), The GRID: Blueprint for a New Computing Infrastructure, Chapter 7, pp. 159–180, 1998
18. A.S. Grimshaw, A. Ferrari and E.A. West, "Mentat", in G.V. Wilson and P. Lu (eds.), Parallel Programming Using C++, Chapter 10, pp. 382–427, 1996
19. F.C. Gartner, "Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments", ACM Computing Surveys, Vol. 31, No. 1, 1999
20. "Access to Knowledge through the Grid in a Mobile World" (AKOGRIMO) Integrated Project FP6-2003-IST-004293. http://www.akogrimo.org/

CrossGrid Integrated Workflow Management System

Martin Maliska, Branislav Simo, and Ladislav Hluchy

Institute of Informatics, Slovak Academy of Science
{martin.maliska, branislav.simo, hluchy.ui}@savba.sk

Abstract. This paper describes a workflow management system that was implemented according to the requirements of a Flood forecasting application [1] developed in the CrossGrid project [2]. The flood forecasting application contains several simulation models (meteorological, hydrological, hydraulic) implemented as the grid jobs. A need for a cascade execution of these models has been a motivation for creating workflow management system capable of executing of a cascade of simulations in the CrossGrid testbed. First implementation of the workflow system has been tied to the portal user interface, but currently has been decoupled as a standalone grid service, which can be used by the two user interfaces developed in the project.

1 Introduction

During the development of the Flood forecasting application[1] in the CrossGrid[2] project we have faced the problem of running a cascade of simulations comprising the application in the project testbed in an automatic way, so that the user will not have to submit each job separately and make the coupling of the input and output files by hand.

The application focuses on the prediction of floods, which are considered as serious problem, causing a lot of damages with many people threatened or even killed. Among the methods that can help to mitigate consequences of floods is a prediction of such an event. Our prediction system consists of several simulation models – meteorological, hydrological and hydraulic, which form the base of a simulation cascade. The cascade is actually a workflow. The meteorological model computes the precipitation in the target area, which is then processed by hydrological model for run-off computation and finally the hydraulic model computes the flow of the water in the river bed and surrounding area. More detailed description can be found in [1]. We are cooperating with experts from Slovak Hydro-Meteorological Institute, which provide us with data for the models and the relevant know-how.

These simulation models are computationally intensive and when several experts will need to run different scenarios, e. g. in the time of possible crisis or parameter studies during model calibration, the computers in the grid will provide the requested processing power.

Our task is to integrate these models so they can be executed as cascade. Each model has several parameters and because every model needs to be calibrated and a calibration has to be carried out for each target area, there should be a way how to easily run specific model(s) or the whole cascade with modified parameters for the same situation (time, space). An expert could also decide to stop the execution of

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1032 – 1039, 2005.

current cascade after he checked the results computed so far and to execute the rest of the cascade with changed parameters or to execute a completely new run of another cascade. These are the key features which affected the design of our application.

In the context of the CrossGrid project, two different user interfaces have been implemented. The first one is a portal interface based on the portlets using Velocity[3] as a template engine. The second one is plug-in for Migrating Desktop – a Java client developed in CrossGrid.

The workflow grid service and user interfaces using it as a back end service are described in the following chapters.

2 Overview of Existing Workflow Management Systems

Several projects use workflows, because they offer them high level abstraction in building the applications. Nowadays, a research is focused on using workflows for execution of grid jobs, web services and grid services.

This research has led to development of various workflow description languages (WDLs). The WDLs have different restrictions for describing a workflow and different features. The projects like Cactus[5], Condor[6] and Unicore[7] use direct acyclic graph (DAG) to describe a workflow. The Unicore has implemented several features to eliminate disadvantages of DAG – it supports conditional and loop execution. An XML based WDL like GJobDL[8], GSFL[9], WSFL[10], SCUFL[11] and their derivation TaskGraph[12] models separately control flow and data flow. Most of them – GSFL, WSFL, SCUFL support nesting of workflow definitions.

The Pegasus[13] is an example of system which produces a concrete workflow from an abstract workflow description. Complete workflow management system P-GRADE[14] based on Condor offers tools for editing, debugging and monitoring of the workflows, performance analysis, distributed checkpointing and dynamic loadbalancing. The Fraunhofer Grid[8] uses Petri-nets for visualisation of a workflow and translates it to the GjobDL and have both explicit and implicit fault management. Another workflow framework for grid application deployment is GridAnt[15] - Apache's Ant based toolkit that specifies taks for compilation, software installation, data transfer and grid applications.

3 Why Another Workflow Management System

As it has been already mentioned, we needed a workflow system that would:

- be able to process our workflows in an interactive way, so that a user could monitor its status and the results during execution,
- not be overly complicated,
- be easily integrated with the middleware developed in CrossGrid and DataGrid[16], especially the job submission service and resource broker,
- support different types of jobs – grid jobs, local jobs to be processed on the server (quick tasks not needing the grid), java tasks to be performed as part of the workflow processing,
- allow integration with the portal user interface.

After reviewing the requirements we have decided to implement the workflow engine by ourselves. The last requirement made us to implement the first version of the engine as a part of the portal. It has worked well but it turned out that such tight coupling with the portal is not scalable and the engine cannot be used by other applications. Therefore we decided to implement it as a standalone grid service using the Globus GT3 toolkit[17]. This implementation is now accessible from both the portal and the workflow plug-in for Migrating Desktop (described later).

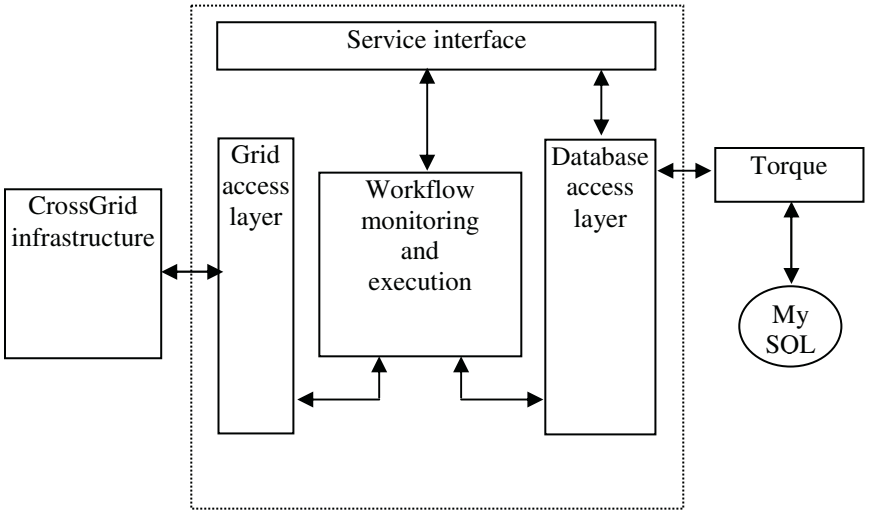


Fig. 1. Workflow service architecture

4 Implementation

The FloodGrid application consists of several simulation models implemented as the grid jobs that are coupled by input and output data. This cascade of models can be easily expressed as a set of activities with exactly defined order of execution – a workflow. A grid service was implemented to provide functionality capable of processing of the workflows as a part of the Flood forecasting application. This service has four modules: service interface, workflow execution and monitoring, grid access layer and database access layer. Figure 1 presents architecture of the service.

Instead of using a workflow description language, we decided to simply store information about the workflow templates, their content and instances to a database.

A workflow consists of several jobs (activities), it is also possible to compose the workflow from several sub-workflows. Dependencies between the jobs (activities) are described by relationships between database tables. The jobs are configurable by input and output parameters. The parameters are bounded to the resources (the files, the directories or the variables). A sharing of resource between several parameters is restricted so only one of the parameters can be an output parameter.

Every job has to have its own class inherited from class *AbstractTask,* which name is stored in database for each job. Inherited class must implement methods for determining two basic states of job – *isAborted()*, *isFinished()* and of course method *run*() that should contains implementation of what job have to do. There is one default implementation of *AbstractTask* used for executing the grid jobs – *GridTask*. This implementation uses a job attribute *param_string* to obtain name of a script file of the job that would be executed in grid.

4.1 Service Interface

The workflow service provides interface which allows these types of operations:

- create, clone, run, remove workflow instance,
- modify job parameters,
- query list of workflows and workflow instances,
- determine current job status,
- basic operations for manipulating with grid proxy certificates.

4.2 Workflow Monitoring and Execution

The workflow service contains two main threads responsible for execution and monitoring of the workflows – a *JobMonitoringThread* and an *ExecutionThread*. While there are multiple instances of the *ExecutionThread* – each running workflow has its own instance, there is only one instance of the *JobMonitoringThread*. The *JobMonitoringThread* is used to monitor state of the grid jobs, other types of jobs are not monitored by it. It has its own queue of running jobs, periodically checks a state of the all jobs from the queue and informs the appropriate *GridTask* object, which belongs to the monitored job. This job object has to decide if any change of a state occurs and whether this change is so important that the *ExecutionThread* owning the job needs to be informed about it. Map of the *ExecutionThreads* searchable by workflow ID is stored in the *JobMonitoringThread*.

An explanation about how the *ExecutionThread* works will be shown on detailed description of steps that are executed when the method *runWorkflow()* is called. At the beginning, tree of the jobs will be created, strictly speaking – tree of the objects that contain implementations of the jobs will be created. Every job object will known all its successors and predecessors and can manipulate with information about the job that belongs to this object stored in the database. Next step is to prepare the root job objects to ready-to-run state and enqueue it to the *ExecutionThread* queue of the job objects with changed state. After the notification about a queue size change, the *ExecutionThread* will wake up, if it sleeps, and process the objects from queue. A decision what to do with the job object will *ExecutionThread* made by determining the state of job. When the job is in state ready-to-run, the job will be executed. Finished job will cause that the job object will be removed from the *JobMonitoringThread* queue and a counter of unfinished jobs will be decreased. Aborted job results in cancellation of the all running jobs according to the workflow. Life cycle of *ExecutionThread* will end when the counter of unfinished jobs will drop to zero value.

4.3 Database Access Layer

All operations that manipulate with information about the workflows stored in database are encapsulated in database access layer. An interface for communication with this layer is provided by *PersistenceLayer* class. It uses object to relational database mapping tool Torque[18] instead of direct access to database. Another mapping mechanism is used for transformation of Torque database objects to objects used by the grid service interface. This mechanism is provided by *DBtoWSMapper* class.

4.4 Grid Access Layer

Grid access layer is an entry point to the CrossGrid infrastructure. A main class, *GridTools,* of this layer supplies methods for submitting a job for execution, getting job status and canceling execution of the job. All these operations are covered by accessing a CrossGrid job submission service. The job submission service uses a job description language (JDL) to specify a grid related information for submitting the job, for example: arguments, input and output sandbox, job type, standard output and input.

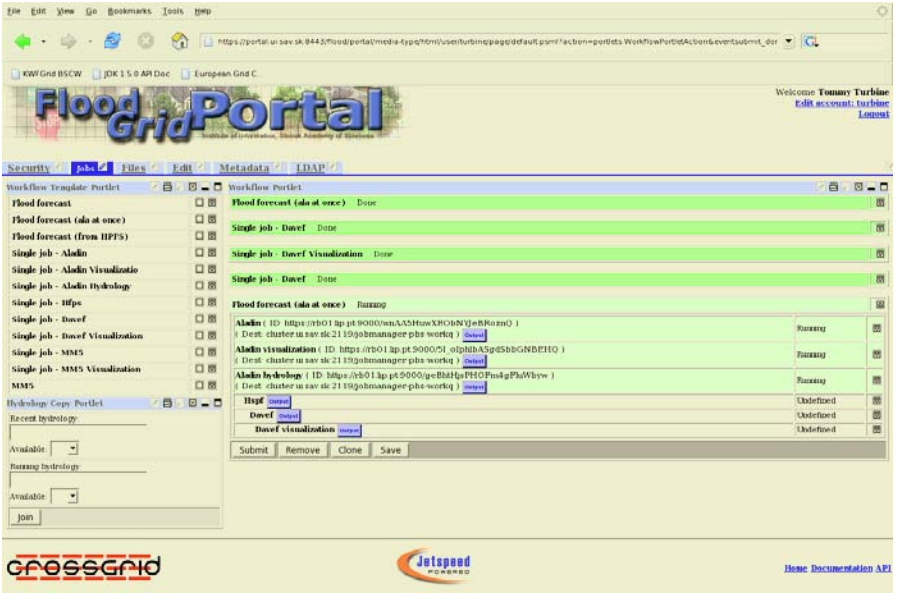


Fig. 2. Screenshot of a portal user interface

5 User Interfaces

During implementation of CrossGrid tools and services, two user interfaces - Portal and Migrating Desktop were developed. According to a need of integration with the

CrossGrid tools, the FloodGrid application was designed to be easily developed with support of both user interfaces. Workflow grid service provides unified interface that these user interfaces uses to manipulate with workflows.

5.1 FloodGrid Application Portal

Portal user interface has been built upon the Jetspeed[4] portal by using Velocity[3] template portlets. The flood application portal contains a portlet for manipulating with proxy certificate, workflow template portlet, workflow portlet, visualization portlet and metadata portlet. Figure 2 captures FloodGrid portal during a work with workflow portlet.

Workflow template portlet displays a list of workflow templates and allows user to select one which a workflow instance will be created from. Created instance and other workflow instances created by user can be monitored by workflow portlet.

This portlet can be used to explore the jobs belonging to a workflow instance and their parameters could be modified. It is possible to view results produced by each job by browsing output directories of the jobs. This functionality is covered by visualization portlet. To reflect a need of application for describing results, metadata support was implemented and its user interface provides metadata portlet, which support this operations: creation, modification, querying.

5.2 FloodGrid Plugin for Migrating Desktop

Migrating Desktop is a java based user interface developed in CrossGrid which emulates desktop environment for grid users. Every application that wants to be integrated to Migrating Desktop has to be implemented as migrating desktop plugin. Functionality of FloodGrid plugin is similar to the FloodGrid portal. Our plugin consists of these panels: workflow template panel, workflow panel, jobs panel, parameters panel and info panel. Workflow template panel and workflow panel operate in similar way as appropriate portlets on the portal except that content of a workflow is shown in a separate panel. Information panel shows relevant information about currently selected object (workflow template, workflow instance, job). Metadata operations are placed on separate tab on tabbed pane.

6 Future Work

We want to focus our effort on making stable version of portal working inside of OGCE[19] and to customize this environment so it will express the needs of expert or common users that will use our application.

Another big task is to improve fault tolerance and upgrade error reporting to the higher level. Cooperation between replica management provided by CrossGrid tools and our metadata management system has to be done so metadata system can work properly.

Although we did not expect a lot of new workflow templates and jobs in our application, it should be nice to have a workflow creation tool. We will decide to adopt an existing one or to develop new one that will fit our needs.

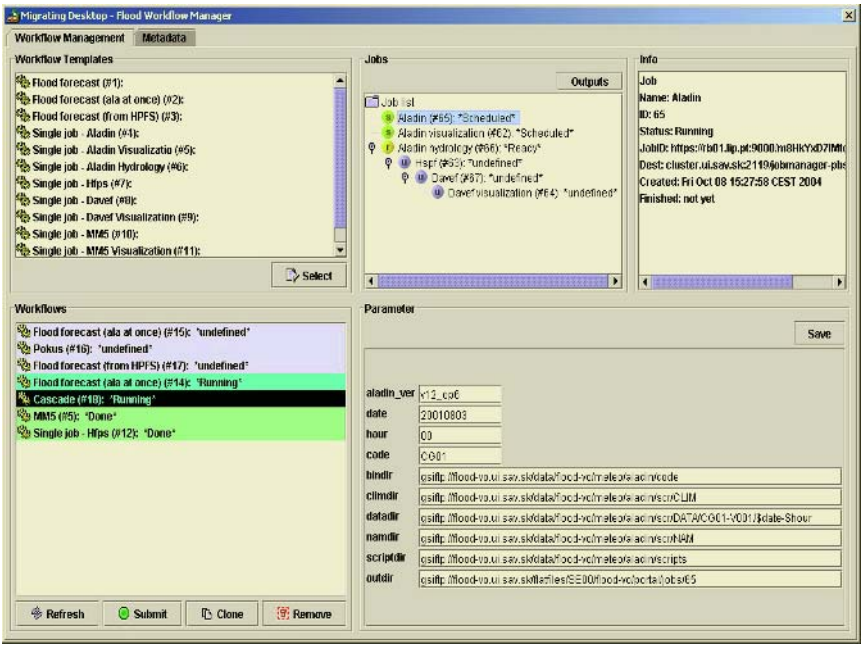


Fig. 3. Migrating Desktop plugin interface

References

[1] Hluchy L., Astalos J., Dobrucky M., Habala O., Simo B., Tran V.D.: Flood Forecasting in a Grid Environment. In: Proc. of 5-th Intl. Conf. on Parallel Processing and Applied Mathematics PPAM'2003, R.Wyrzykowski et.al. eds., 2004, LNCS 3019, Springer-Verlag, pp. 831-839, ISSN 0302-9743, ISBN 3-540-21946-3. September 2003, Czestochowa, Poland.

[2] CrossGrid - Development of Grid Environment for Interactive Applications. IST-2001-32243. http://www.crossgrid.org (visited November, 2004)

[3] Velocity Template Engine, http://jakarta.apache.org/velocity/, (visited November, 2004)

[4] Jetspeed portal framework, http://portals.apache.org/jetspeed-1/, (visited November, 2004)

[5] Cactus project, www.cactuscode.org, (visited November, 2004)

[6] The Condor project, www.cs.wisc.edu/condor/, (visited November, 2004)

[7] The UNICORE project, http://unicore.sourceforge.net/ajo.html, (visited November, 2004)

[8] Hoheisel, A., Der, U.: An XML-based Framework for Loosely Coupled Applications on Grid Environments. In: P.M.A. Sloot et al. (Eds.): ICCS 2003. LNCS 2657, Springer-Verlag, pp. 245–254

[9] S.Krishnan, P.Wagstrom, G.von Laszewski GSFL: A Workflow Framework for Grid Services

[10] Technology Report: Web Services Flow Language (WSFL), http://xml.coverpages.org/wsfl.htm, (visited November, 2004)

[11] MyGrid project homepage workflow section, http://twiki.mygrid.org.uk/twiki/bin/view/Mygrid/WorkFlow#XScufl_workflow_definition,
[12] Triana project, http://www.triana.co.uk/, (visited November, 2004)
[13] Planning for Execution in Grids, http://pegasus.isi.edu/, (visited November, 2004)
[14] P_GRADE, http://www.lpds.sztaki.hu/pgrade/, (visited November, 2004)
[15] The GridAnt, http://www-unix.globus.org/cog/projects/gridant/, (visited November, 2004)
[16] The DataGrid project, http://eu-datagrid.web.cern.ch/eu-datagrid/, (visited November, 2004)
[17] The Globus Alliance, http://www.globus.org/, (visited November, 2004)
[18] Torque, http://db.apache.org/torque/, (visited November, 2004)
[19] Open Grid Computing Environment (OGCE), http://www.collab-ogce.org/nmi/index.jsp, (visited November, 2004)

Load Balancing by Changing the Graph Connectivity on Heterogeneous Clusters

Kalyani Munasinghe[1,2] and Richard Wait[2]

[1] Dept. of Computer Science,
University of Ruhuna, Sri Lanka
[2] Dept. of Information Technology,
Uppsala University, Sweden
{kalmun, richard}@it.uu.se

Abstract. This paper examines the problem of adapting parallel applications on a cluster of workstations. The cluster is assumed to be a heterogeneous, multi-user computing environment so that efficient load balancing within the application must take external factors into account. At any time the users of the network are competing for resources. Performance of a particular processor, as a component in the parallel (message passing) computation, depends on both static factors, such as the processor hardware, and dynamic factors, such as the system load and the activities of other users. For each processor, the external factors can be condensed into a single parameter, the load index, which is a normalised measure of the current spare capacity of the processor available to the application.

Numerical experiments show the efficiency of the load balancing strategies on a finite element application with a domain decomposition and the effect on overall computation time.

1 Introduction

Shared cluster networks provide a useful platform for parallel applications because of their cost performance ratio. The cluster environment can offer high performance if resources are managed efficiently. One of the problems in achieving high performance in clusters is that resources may not be fully under control of the individual application. In this environment, parallel programs may be competing for resources with other programs and may be subject to resource fluctuation during execution. In such a system, an important issue is to find effective techniques that distribute the tasks of a parallel program appropriately on processors. One problem is how to schedule the tasks among processors to achieve goals such as minimizing execution time or maximizing resource utilization.

For example, an irregular finite element mesh, may be partitioned into subdomains and each subdomain assigned to a single processor. Assuming that the

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1040–1047, 2005.

computational effect is proportional to the size of the subdomain, two questions arise:

1. What is the optimal size for each subdomain?
2. How can the problem domain be partitioned into such subdomains?

On a homogeneous cluster of dedicated processors (e.g. Beowulf [2]) with a fixed problem size, the partition may be uniform and static.

For some irregular grid applications, the computational structure of the problem changes from one computational phase to another. For example, in an adaptive mesh, areas of the original graph are refined in order to model the problem accurately. This can lead to a highly localized load imbalance in subdomain sizes. Alternatively, load imbalance may arise due to variation of computational resources. For example in a shared network of workstations, computing power available for parallel applications is dynamically changing. The reasons may be that the speed of machines are different or there are other users on some part of the cluster, possibility with higher priority. The partitioning has to be altered to get a balanced load. We propose an algorithm which reduces the load imbalance by local adjustments of current loads to reduce the load spike as quickly as possible and to achieve a load balance. It is assumed that the connections for data transfers between the processors are determined by the data locality but data movement should be kept as low as possible. The load balance is adjusted in general by migrating data to adjacent processors with modifications to the connectivity where necessary.

2 Background

2.1 Some Definitions

Let p be the number of processors. The processor graph is represented by a graph (V, E) with $|V| = p$ vertices and $|E|$ edges. Two vertices i and j form an edge if processors i and j share a boundary of the partitioning. Hence the processor graph is defined by the topology of the data subdomains. As the edges of the processor graph are defined by the partitioning of the domain, when the partition changes the graph topology may change. Each vertex i is associated with a scalar l_i, which represents the load on the processor i.
The total load is

$$L = \sum_{i=1}^{p} l_i \tag{1}$$

The average load per processor is

$$\bar{l} = \frac{1}{p} L \tag{2}$$

and we can define the vector, b, of load imbalances as

$$b_i = l_i - \bar{l} \tag{3}$$

This definition is based on the assumption that in order to achieve a perfect balanced computation, all the loads should be equal. If however the processor environments are heterogeneous and corresponding to each processor there is a load index α_i which can be computed using current system and/or processor information, then the ideal load $\tilde{l_i}$ is defined as

$$\tilde{l_i} = \alpha_i \frac{1}{\sum_j \alpha_j} L \tag{4}$$

Load difference from the ideal load can be defined as

$$d_i = l_i - \tilde{l_i} \tag{5}$$

A processor is therefore overloaded if $d_i > 0$. These simple definitions assume that the computation can be broken down into a large number of small tasks each of which can be performed on any processor for the same computational cost. This is not necessarily true as for example in a finite element computation, the cost of the computation might depend on the number of edges between subdomains in addition to the cost proportional to the size of the subdomains. So the total distributed computational cost is not necessarily equal after any redistribution of the data.

A processor is highly overloaded if the load difference is excessive

$$d_i > cl_i (\text{for some constant } c < 1)$$

typically $c \approx 0.3$ was used in the experiments, a partition is balanced if no processor is overloaded.

In order to reduce any unnecessary fragmentation of data, data will in general only be moved between contiguous subdomains. It is assumed that any processor is equally accessible from all other processors.

3 Related Work

Different techniques have been proposed for adapting parallel applications running on clusters. Different dynamic load balancing and migration strategies have been proposed.

There are many studies dealing with the problem of load balancing for distributed memory systems. Some work [6] assume that the processors involved are continuously lightly loaded, but commonly the load on a workstation varies in an unpredictable manner.

There are algorithms exist for scheduling parallel tasks. The Distributed Self Scheduling (DSS) [7] technique uses a combination of static and dynamic scheduling. During the initial static scheduling phase, p chunks of work are assigned to the p processors in the system. The first processor to finish executing its tasks from the static scheduling phase designates itself as the centralized processor and it stores the information about which tasks are yet to be executed, which

processors are idle and dynamically distributes the tasks to the processors as they become idle.

Alessandro [3] introduced a method to obtain load balancing through data assignment on a heterogeneous cluster of workstations. This method is based on modified manager-workers model and achieves workload balancing by maximizing the useful CPU time for all the processes involved.

Dynamic load balancing scheme for distributed systems [5] considers the heterogeneity of processors by generating a relative performance weight for each processor. When distributing the workload among processors, the load is balanced proportional to these weights.

The AppLes approach [1] uses parameterizable application and system specific models to predict application performance using a given set of resources. Using these models and forecasts of expected resource load, an AppLeS agent selects a resource set and an application schedule by evaluating candidate mappings. The mapping with the best expected performance is implemented on the target resource management system.

Many methods proposed in the literature to solve the load balancing problem are applicable to adaptive mesh computation. One of the earliest schemes was an iterative diffusion algorithm [4]. At each iteration, new load is calculated by combining the original load and the load of neighbouring processors. The advantage of this approach is, it requires local communication only, but the problem is its slow convergence. Several scratch-remap [8] and diffusion based [9] adaptive partitioning techniques have also been proposed. These different approaches are better for different system environments and different computational environments. In our approach, we try to identify sharp increases and to reduce them quickly as possible without necessarily achieving a perfect load balance.

4 Repartitioning with Minimum Data Movement

In this section, we describe our proposed approach. It operates on the processor graph which describes the interconnection of the subdomains of a mesh that has been partitioned and distributed among the processors.

We assume that an overloaded node initiates the load balancing operation whenever it detects that it is overloaded.

An important feature of our approach is to capture the need for the processor load to adapt very quickly to external factors for example, a key press or a mouse click may indicate that machine is no longer available. This is useful assuming we can use workstations only if the owner not using it and we need to move load when ever the owner returns. A subdomain is then deleted and the corresponding processor emptied of all load. If at a later time, the processor becomes available again a new subdomain may be created, possibly in another part of the graph.

If a processor is to be removed then the load has to be distributed onto neighbouring processors that are lightly loaded. The neighbouring processors are defined by the subdomain connectivities. The load to be distributed of partitioned into sections that are proportional to the load differences d_i of the neighbours

that are not already overloaded. The redistribution has two phases, the data partition and the data movement. The partitioning uses a greedy algorithm. In a typical finite element computation with an unstructured mesh distributed as subdomains, the partitioning starts from the subdomain boundaries adjacent to the lightly loaded neighbours and reallocates the old subdomain into appropriately sized sections. The mesh data is then transferred to the new subdomains, the processor connectivities are modified to take account of the new subdomain topology and the processor is released.

Those processors that are overloaded to a lesser degree, i.e. that need to shed some load but will remain as part of the computational cluster with a nontrivial load after the redistribution, also redistribute load to their lightly loaded neighbours using a similar greedy approach selected parts of the subdomain to be redistributed starting from the boundaries. These modifications may also result in changes to the subdomain topology and hence to the processor connectivities.

Additional processors, when available, may be (re)introduced at the point in the processor graph were the data movement is greatest.

5 Experimental Results

The experiments were performed on the problem of using the finite element method on an unstructured grid. Here we assumed that the computation is element based so that the load to be redistributed can be considered as repartitioning of the elements into subdomains, i.e. partitioning the dual graph.

Our proposed algorithm was implemented in *C* and *MPI* on 8 *Sun* workstations connected by 100 Mb/s Ethernet. All *Sun*s share a common file server and all files are equally accessible from each host due to the implemented NFS (Network File System). *Unix* provides a large amount of statistical information that can be used to describe a workload. Here we used a simple load sensor which uses *Unix* commands to collect the system information. The load sensor calculates percentage of unused CPU of each machine. Here we used a combination of processor speed and unused CPU amount as a load index. For loosely coupled linux clusters that do not incorporate NFS it same results can be gathered us-

Table 1. System Information

Processor	Speed Mb/s	Unused CPU	Load Index
1	300	99	29700
2	360	92	33120
3	333	99	32967
4	450	70	31500
5	333	99	32967
6	300	99	29700
7	450	71	31950
8	333	98	32634

Table 2. Initial Distribution

Processor	1	2	3	4	5	6	7	8
Load	540	540	604	540	444	617	540	495

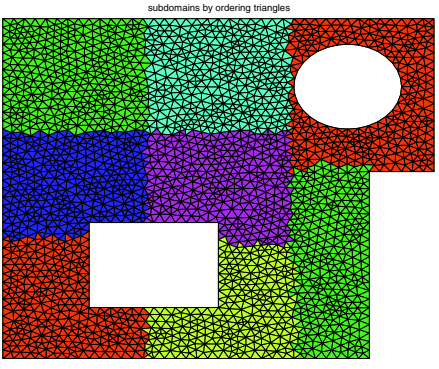


Fig. 1. Original Grid

Table 3. Test1: Running Times in milliseconds

Initial Partition	With Load Balance
16.7276	
	iteration 1 2.51483
	iteration 2 2.35808
	iteration 3 2.31213

ing software agents. In our environment, some machines can only be used if the owner is not using it and the processes should be moved if the user returns before they finish. If a mouse is moved or a key pressed, we need to move application workload from that particular machine and this information overides the normal load index. In order to determine when such a machine can be returned to the cluster, it is necessary to identify inactive time of a machine, this is achieved by a simple script in the background which gives the idle time of the machine.

The table 4 gives a typical snapshot of the system information on each machine in the cluster used in the experiments.

The experiment illustrated is a small finite element calculation, the initial partitioning of the grid into subdomains is shown in figure 1. The sizes of the subdomains are shown in table 5, the mean size is 540.

The finite element solution was computed iteratively, the times given in table 5 are for one iteration with the initial distribution and the first three iterations with a load balancing step between each iteration.

In the second experiment, the initial load was modified so that it was more unbalanced an the results shown in figure 2 illustrate how quickly the load on

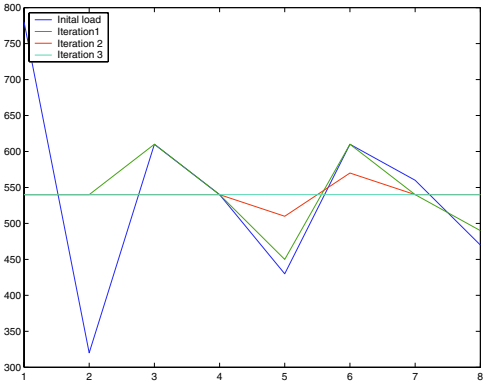


Fig. 2. Test2:Reduction of Load Spike

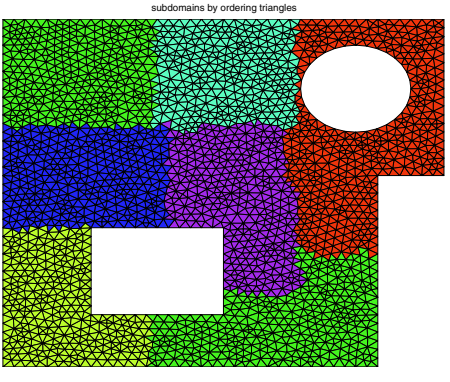


Fig. 3. Test3: Grid after removing one processor

Table 4. Test3: Redistribution of load

Processor	1	2	3	4	5	6	7	8
Load	737	618	618	619	539	588	601	0

a heavily overloaded node is reduced, again a single load balancing step was allowed after each iteration.

The results of the third experiment, in figure 3, show the redistribution of the load if one processor is removed, the distribution of the load is given in table 5 shows how the load is removed from the processor that is no longer available, it is not necessarily distributed evenly as the load indices of the machines may vary.

In the fourth test, the additional processor was reintroduced after several iterations when the load had become evenly balanced between the other processors (assuming equal load indices in this case).

Table 5. Test4: Reintroduction a processor

Processor	1	2	3	4	5	6	7	8
Iteration 1	618	618	617	617	629	618	603	0
Iteration 2	618	540	617	540	540	540	540	385
Iteration 3	540	540	540	540	540	540	540	540

Table 6. Different Load Index

Load Index	Run time
Processor speed x Unused CPU	2.51483
Unused CPU	6.2142

The final results in table 5 illustrate how the timings depend on the choice of load index.

6 Conclusions

Assuming that a single overloaded node has a greater effect on overall efficiency than a single underloaded node, we have presented an approach to load balancing that attempts to reduce an imbalance due to a load spike as quickly as possible. The experimental results show a performance improvement with the approach. According to the experimental results, we can see that the load index also plays an important role. Our future work includes experimenting on larger clusters with larger data sets.

References

1. `http://www.-cse.ucsd.edu/users/breman/apples.html/`.
2. `http://www.beowulf.org/`.
3. Alessandro Bevilacqua, *A dynamic load balancing method on a heterogeneous cluster of workstations*, Informatica **23** (1999), no. 1, 49–56.
4. G. Cybenko, *Dynamic load balancing for distributed memory multiprocessors*, Parallel and Distributed Computing **7** (1989), 279–301.
5. Zhilling Lan and Valerie E. Taylor, *Dynamic load balancing of SAMR applications on distributed systems*, Scientific Programming **10** (2002), 319–328, no. 21.
6. C. K. Lee and M. Hamdi, *Parallel image processing application on a network of distributed workstations*, Parallel Computing **26** (1995), 137–160.
7. J. Lin and V. A. Saletore, *Self scheduling on distributed memory machines*, Super-Computing (1993), 814–823.
8. L. Oliker and R. Biswas, *Plum: Parallel load balancing for adaptive structured meshes*, Parallel and Distributed Computing **52** (1998), no. 2, 150–177.
9. Kirk Schloegel, George Karypis, and Vipin Kumar, *Multilevel diffusion schemes for repartitioning of adaptive meshes*, Journal of Parallel and Distributed Computing **47** (1997), no. 2, 109–124.

Threat Model for Grid Security Services*

Syed Naqvi and Michel Riguidel

Graduate School of Telecommunications,
ENST, 46 Rue Barrault, 75634 Paris Cedex 13, France
{naqvi, riguidel}@enst.fr

Abstract. The grid computing paradigm involves both the availability of abundant computing resources, and the storage of increased amounts of valuable data. Such information systems heavily rely upon the provision of adequate security. It is imperative that techniques be developed to assure the trustworthiness of these environments. Formal verification provides the tools and techniques to assess whether systems are indeed trustworthy, and is an established approach for security assurance. When using formal verification for security assessment one of the most important concerns should be to be precise about the threat model. A comprehensive threat model is indispensable for the simulations of a grid security model. This article presents a survey of the various threat models and discusses how and when these threat models may be inappropriate for use in the grid computing environments. Then a fine-grained threat model for grid computing is presented.

1 Introduction

The report of a survey conducted by the Computer Science Department of Virginia Tech among members of the grid community [1] states that more than half of the respondents believe that existing grid security solutions do not provide adequate services for collaborative grid communities. The reasons given ranged from the *lack of an underlying threat model* to the complexity and expense of inter-site trust relationships that are currently required.

A *threat model* is used to describe a given threat and the harm it could do a system if it has a vulnerability; whereas a 'threat vector' is the method a threat uses to get to the target. The construction of the threat model is an essential and critical phase in the construction of the overall security model for a system; the threat model is one of the factors that feeds into the security model, which then takes and analyses the costs and

* This research is supported by the European Commission funded project SEINIT (Security Expert Initiative) under reference number IST-2002-001929-SEINIT. The overall objective of the SEINIT project is to ensure a trusted security framework for ubiquitous environments, working across multiple devices and heterogeneous networks, in a way that is organization independent (inter-operable) and centered around an end-user. Project webpage is located at www.seinit.org.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1048 – 1055, 2005.

benefits of defending against each listed threat. In the evolution of computational grids, security threats were overlooked in the desire to implement a high performance distributed computational system. So far, the grid technology has been little used except by a certain kind of public (mainly academics and government researchers). This public benefit greatly from being able to share resources on the grid, and have no intention of harming the resource owners or fellow users. This is all about to change. The number of people who know about the grid is growing fast, as are the worthwhile targets for the potential attackers.

The growing size and profile of the grid require comprehensive security solutions as they are critical to the success of the endeavor. A comprehensive security system, capable of responding to any attack on grid resources, is indispensable to guarantee its anticipated adoption by both the users and the resource providers. The conception of a comprehensive security model for the grid requires a realistic threat model. Without such a threat model, security designers risk wasting time and effort implementing safeguards that do not address any realistic threat to the grid. Or, just as dangerously, they run the risk of concentrating their security measures on one threat while leaving the grid architecture dangerously exposed to others.

This paper is organized in the following manner: an overview of the grid-specific security requirements is presented in section 2. A concise description of the related work is given in section 3. Our proposed threat model is elaborated in section 4. Finally, some conclusions are drawn in section 5 along with an account of our future directions.

2 Grid-Specific Security Requirements

Current Grids provide a consistent set of security services based on X.509, PKI, or Kerberos authentication, proxy certificates to carry the user authentication to remote resources, and a set of secure communication primitives based on the IETF GSS-API: secure telnet, remote shell, and secure ftp are provided by using these services. However, the scope of the computations and the data spread across the Grid require an in-depth security mechanism to effectively handle the Grid-specific security requirements. A concise account of these Grid-specific security requirements, identified in [2], are given below:

2.1 Authentication

The main requirements for authentication include scalability, trust across different Certification Authorities and timeliness of revocation. Although the technology for authentication is quite well-established, it is still not matured to be integrated into emergent Grid environments.

2.2 Authorization

In a Grid context, the potentially large numbers of users and resources with differing management and policies does not permit the use of *general access rights*. Scalability

becomes an important factor when considering authorization mechanisms within a Grid environment, especially due to its dynamic nature.

2.3 Revocation

Revocation is vital for both authentication and authorization. For a Grid to be trustworthy it must support an instant withdrawal of access rights. Such dynamic mechanism of revocation is an important part of manageability.

2.4 Confidentiality

Confidentiality within a Grid is not just concerned with data that is stored upon a resource; it also extends to the privacy requirements of the actual users and resources, to protect the deducibility of data, and to ensure consistence of confidentiality in data replication process within the Grid.

2.5 Distributed Trust

Distributed trust is also closely bound up with but broader than authorisation. Since the driving application of many grids requires users to be able to compile and run arbitrary code, this ability is built into that trust model. The trust status of a remote system or user is hard to determine so is the communications medium. Yet a grid must be constructed from such components, in a dynamic fashion.

2.6 Integrity

The data from some projects may not be confidential, but huge datasets from very expensive experiments cannot easily be re-generated. They must be beyond reproach. This becomes a grid problem when copies or subsections of those data are managed automatically and stored at dispersed, separately managed locations. Integrity requirements also extend to the mechanisms by which users' rights are delegated.

2.7 Non-repudiation

Non-repudiation is not a security aspect that has been considered in any detail within the grid, but it will become important as the grid technology matures and is used by applications involving financial exchanges. If accessing resources starts costing money, then both parties must be assured that the other fulfils their duty; in the cases where one party fails, proof of commitment is necessary to formally resolve the dispute.

Beside the abovementioned security services, a comprehensive security mechanism for the Grid should also include some reliable means of providing abstraction of the underlying security services.

3 Related Work

To our knowledge, there is no threat model that addresses the grid-specific threats paradigm. In the report of the Accelerated Trustworthy Internetworking Workshop

[3], the Trusted Computing Group remarked that "Threat models are missing – leading to inappropriate application of security mechanisms."

UK e-Science sponsored project DAME (Distributed Aircraft Maintenance Environment) [4] has a security workgroup which is working to develop threat models for the DAME system. They are studying established software dependability techniques and seeking to establish a methodology for dependability analysis in distributed Grid systems. Their long-term objective is to identify the main threats and risks in deployed systems such as DAME, and to access the efficiency of the OGSA/OGSI security models for managing these risks.

The project SHARP (Secure High Available Resource Peering) [5] defines a threat model that is limited to authentication and authorization. Other features such as confidentiality are not addressed.

The Denali project [6] of the University of Washington assumes a light threat model with no consideration of the covert channels between virtual machines.

Some threat models [7, 8, 9] for P2P systems have been developed; however, they can not be directly applied to the grids, as they do not address a number of grid-specific threats which arise from the resource sharing phenomenon of the grids.

4 The Proposed Threat Model

Various threats covered in this model can be broadly classified as:

1. Threats to the grid resources when the data/applications are in the non-critical state.
2. Threats to the grid resources when the data/applications are in the critical state.
3. Threats from the grid.

Some threats to the grid resources remains ostensibly the same regardless of data/applications state (critical or not); however, their nature and extent widely varies. These threats are elaborated in sections 4.1 and 4.2. The threats from the grid pose risks to the society – this is a social problem that has to be addressed to protect the grid resources from being misused by the ill-intention people who may use the enormous computing power for malicious purposes. The threats from the grid resources are detailed in section 4.3.

4.1 Threats to the Grid Resources When the Data/Applications are in the Non-critical State

A certain number of threats are always posed to the grid resources even if the data and/or applications are in the non-critical state (i.e. the data across these resources are not being exchanged and there is no application running over it). These static threats are discussed in this section.

4.1.1 Threats to Integrity

These threats are directly related to the physical infrastructure as the integrity of the replica files and other stored data is dependent on the integrity of the grid hardware.

The physical infrastructure of a grid is not confined in a single administrative domain where some centralized intrusion detection mechanism may be employed. The grid resources are spread over the countries and hence they have multiple threats of physical intrusion by malicious intruders, accidents like fire, short-circuits or electrical surges, natural hazards such as earthquakes and floods.

4.1.2 Threats to Confidentiality

Threats of viruses, worms, and trojan horses have serious implications for the grid resources as their broader scope and the extent of damage make them more attractive to developers of the malicious codes. Apart from these threats, the unauthorized disclosure of information without changing the state of the system is a serious threat to the confidentiality because it is hard to detect. It would not result in any modification to any information contained in the grid resources as neither the operation nor the state of the system is changed.

4.1.3 Threats to Availability

These include threats of preventing or delaying authorized access to grid resources. In the time-critical applications any delay or denial of access to the services inflicts heavy losses and that is why the availability of computing resources, even if there is no application running, is crucial. Other threats to availability are the failure of computing resources and the power interruption.

4.1.4 Threats to Access Control

If the authentication and authorization mecha- nisms of a grid are not strong enough to properly handle the access control then unauthorized persons can get access to its resources that will put the overall security of the grid architecture at stake.

4.2 Threats to the Grid Resources when the Data/Applications Are in the Critical State

The threats to a grid are more severe when the data and/or applications are in the critical state (i.e. applications are running over it and data being exchanged across its resources). These threats are discussed in this section.

4.2.1 Threats to Integrity

These include alteration of grid data and threats to robust- ness. A malicious subject may gets involved in the grid applications in order to destroy or replace the actual data on its resources. The grid applications are data-critical applications and any misadventure to the data will cause havoc to their functioning. Threats to robustness target the replica files and backup servers so as to interrupt the smooth running of the grid applications in the case of any anticipated failure of a grid component.

4.2.2 Threats to Confidentiality

These threats include eavesdropping and masque-rading. The potential targets of these threats are temporary files and transit data items. A number of shared resources in a grid provides a fertile ground for launching such attacks from a weak node. In these

threats a malicious subject tries to silently observe the resources to obtain classified data/information. The scope of the applications running over a grid may require absolute confidentiality especially in the situations where privacy violation is irreversible like medical applications.

4.2.3 Threats to Availability

These threats include denial of service (DoS) and buffer overflow. They result in the prevention of authorized access to resources or the delaying of time-critical operations. The availability of grid resources is severely threatened by the distributed denial of service (DDoS) where the attack begins by exploiting a vulnerability in one computer system and making it the DDoS *master*. It is from the master system that the intruder identifies and communicates with other systems that can be compromised.

4.2.4 Threats to Communications

These are the threats to the transportation of data across the grid resources. The presence of security gaps exacerbate the security of the communication mediums. The security gaps are introduced in any secure path going through one or more middleboxes that need to perform some processing on passing data packets. These middleboxes include Network Address Translation (NAT) gateways, packet or content filters, proxy firewalls, and Wireless Application Protocol (WAP) gateways. In the grid context, security gaps could surface, particularly in cases where some grid resources and nodes exist in a local network behind a firewall. Another threat to communication channels is the use of passive wire tapping to observe information being transmitted over a communications line.

4.3 Threats from the Grid Resources

The enormous amount of computing power of computational grids may be exploited by the ill-intention people for their malicious designs. The results of such computations will be used against the society and hence there is a strong need to counter these menaces. Unlike the *accidental threats* that exist with no premeditated intent, these threats from the grid are the *intentional threats* that exist with deliberate intentions.

One of the planned uses of the new tera scale grid [10] is to simulate terrorist attacks to help government agencies prepare for worst-case scenarios [11]. However, this is the point of great concern because the same amount of resources can be used by the terrorists to simulate their designs. Moreover, other illicit activities like cloning etc. can be flourished by using the abundant computing powers of grid. These threats from the grid resources to the society can neither be ignored nor their exploitation be ruled out.

4.4 Simulations of the Threat Model

A wide variety of simulating tools for the grid have been developed around the world. They include OptorSim [12], ChicagoSim [13], SimGrid [14], GridSim [15], EDGSim [16], GridNet [17] etc. However, the available range of these simulators does not provide any support for the simulations of grid security functionalities. This

situation has obliged us to first develop modules in a programming language to perform grid security simulations. We are currently working on the development of simulation modules to evaluate the performance of grid security functionalities by using our proposed threat model.

5 Conclusions and Future Plans

Developers and users should be aware of the main threats in each area of grid security, and should develop, deploy and use grid technology in such a way that *security in depth* is established as far as possible.

It is important to remember that security is a process, the threat picture is always changing, and threat analysis needs to be continuously updated. In other words, grid infrastructure should be subject to constant review and upgrade, so that any security loophole can be plugged as soon as it is discovered. The growth in the grid community should lead to improvements as larger number of users will find the loopholes faster, and more developers will be available to fix them and release patches.

We are currently working on the development of simulation modules to evaluate the performance of grid security functionalities by using our proposed threat model. We shall formulate scenarios for the threats mentioned in this model so that they can be simulated and the performance of the model be evaluated. Finally, based on the simulation results, necessary changes will be made in the security threats model.

References

1. Lorch M., Kafura D., *Grid Community Characteristics and their Relation to Grid Security*, Technical Report TR-03-20, Computer Science, Virginia Tech., June 2003
2. Broadfoot P. and Martin A., *A Critical Survey of Grid Security Requirements and Technologies*, Technical Report PRG-RR-03-15, Oxford University Computing Laboratory, August 2003.
3. Final Conference Report, The Accelerating Trustworthy Internetworking Workshop (ATI2004), April 2004 http://gtisc.gatech.edu/ati2004/ATI_Report_FINAL_4-25-04.pdf
4. Jackson T., Austin J., Fletcher M., Jessop M., *Delivering a Grid enabled Distributed Aircraft Maintenance Environment (DAME)*, Proceedings of UK e-Science All Hands Meeting 2003 (AHM2003), Nottingham, UK, September 02-04, 2003
5. Fu Y., Chase J., Chun., Schwab S., and Vahdat A., *SHARP: An Architecture for Secure Resource Peering*, Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, August 2003
6. Whitaker A., Shaw M., and Gribble S., *Denali: Lightweight Virtual Machines for Distributed and Networked Applications*, University of Washington Technical Report February 02, 2001
7. The Cascade Project – Media Networks Laboratory, Department of Computer Science, Stony Brook University http://www.mnl.cs.sunysb.edu/project/cascade
8. DeFigueiredo D., Garcia A., and Kramer B., *Analysis of Peer-to-Peer Network Security using Gnutella*, University of California Report, December 2002

9. Condie T., Kamvar S., Garcia-Molina H., *Adaptive Peer-to-Peer Topologies*, Proceedings of the Fourth IEEE International Conference on Peer-to-Peer Computing, Zurich, Switzerland, August 25-27, 2004
10. The TeraGrid Project – http://www.teragrid.org
11. Shread P., *New Terascale Grid to Simulate Terrorist Attacks*, Grid Computing Planet, June 12, 2002 http://www.gridcomputingplanet.com/news/article.php/3281_1365171
12. Cameron D., Carvajal-Schiaffino R., Millar P., Nicholson C., Stockinger K., and Zini F., *OptorSim: A Grid Simulator for Replica Optimisation*, UK e-Science All Hands Conference 31 August - 3 September 2004.
13. Ranganathan K. and Foster I., *Identifying Dynamic Replication Strategies for a High Performance Data Grid*, Proceedings of the International Grid Computing Workshop, Denver, Colorado, USA, November 2001.
14. Legrand A., Marchal L., Casanova H., Scheduling Distributed Applications: The SimGrid Simulation Framework, Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid 2003 (CCGrid2003), May 12-15, 2003, pp 138-145
15. Buyya R. and Murshed M., *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, The Journal of Concurrency and Computation: Practice and Experience, Wiley Press, May 2002. pp 1-32
16. EDGSim: A Simulation of the European DataGrid. http://www.hep.ucl.ac.uk/~pac/EDGSim
17. Lamehamedi H., Shentu Z., Szymanski B., and Deelman E., *Simulation of Dynamic Data Replication Strategies in Data Grids*. Proceedings of the International Parallel and Distributed Processing Symposium 2003 (IPDPS2003), April 22-26, 2003

A Loosely Coupled Application Model for Grids

Fei Wu and K.W. Ng

Dept. of Computer Science and Engineering,
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{fwu, kwng}@cse.cuhk.edu.hk

Abstract. Scheduling distributed applications effectively and efficiently on Grid environments is difficult because of the dynamic and heterogeneous characteristics of the Internet. In this paper, we propose a loosely coupled application model for building distributed applications on Grids. We assume that a Grid application is composed of a group of independent modules. Each module performs either a remote service request or local processing. Different modules in such an application exchange information by explicitly described data that can be understood by both the application and the Grid environment. Each module is triggered by its input data, and finally it produces some output data. All information exchanges are completed transparently as they are carried out by the Grid management system. We call a module in such an application a loosely coupled module (LCM). A loosely coupled application can be defined by the combination of dependent or independent LCMs. By the loosely coupled application model, Grid applications can be built by employing discrete and heterogeneous resources on the Internet. The loosely coupled relationships among different LCMs can guarantee the robustness of the application. Parameters are defined in the application model so that application schedulers in the Grid environment can efficiently implement application scheduling by designing appropriate scheduling algorithms based on these parameters.

1 Introduction

The main goal of Grid computing [1] [2] [3] [4] is to effectively organize various computational resources distributed on the Internet to provide computing facilities to users as a large virtual computer. A Grid application can dynamically compose a large number of resources across the environment and implement its computations with high performance. The two most important features that distinguish a Grid from traditional distributed systems are heterogeneous resources and the dynamic network. Traditional distributed and parallel applications are hard to be scheduled on Grid environments because they presume a homogeneous and stable execution environment. The ease of development of Grid applications is a key problem to make the Grid a mature platform for general-purpose computing. While many studies on the development of Grid applications have been put forward, a common opinion is that current tools and languages are insufficient to develop effective and efficient applications for the Grid environment. Many issues must be tackled to bridge the gap between Grid applications and Grid environments, such as interoperability, adaptability,

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1056 – 1065, 2005.

service discovery, application performance, large-scale data transfer, robustness, security, schedulability, etc. [5] [6]. Among these issues, robustness, adaptability and schedulability are deemed to be especially important because they promise the validity of Grid applications and guarantee the availability and performance of Grid environments.

In this paper, we propose a loosely coupled application model to guarantee the robustness, adaptability and schedulability of Grid applications. A loosely coupled application is composed of some independent software components and the corresponding data set that the components will process. Different components exchange information by explicitly described data in the data set. When running on a Grid, different components of such applications can be scheduled flexibly according to the runtime status of Grid resources. In this application model, the robustness of a Grid application can be guaranteed from two aspects: applications are composed in a loosely coupled style so as to reduce the effect on the whole applications when a partial error occurs; the necessary knowledge about Grid applications are known by the environment so that remedial actions can be applied to ensure applications can be executed correctly. As the modules in such loosely coupled applications are more independent than in tradition manners, adaptability and schedulability of these applications are much increased. The rest of this paper is organized as follows: Section 2 gives the definitions of loosely coupled applications and their properties; Section 3 describes the scheduling problems of loosely coupled applications; some future work will be outlined in Section 4, and finally, a conclusion of this work will be given.

2 The Loosely Coupled Application Model

2.1 Loosely Coupled Applications

A Grid application is a distributed application consisting of a number of components that runs in a Grid environment. The dependencies between different components can be an important factor that influences the performance of the application. In Grid environments, if the dependencies among components are weak, any partial failure will produce a smaller influence on the whole execution of the application. A strong dependency increases the probability of application failures and at the same time, the communication load. For distributed applications, the dependencies can be represented by data exchange. Suppose that *A* and *B* are two related modules in an application. If module *A* and module *B* know the internal information of each other, they can exchange data by data sharing or synchronized message passing. We call such relationship tightly coupled. The message passing model is a typical example of this class. If module *A* knows *B*'s interface and they communicate by asynchronous messages, we call it moderately coupled. Some implementations of distributed objects and Web services fall into this style. If modules *A* and *B* do not know each other, and their produced and required data are coordinated by a third-party unit or system, we say that the two components are loosely coupled. The messages in such a situation are called loosely coupled messages. These three kinds of relationships are shown in Figure 1.

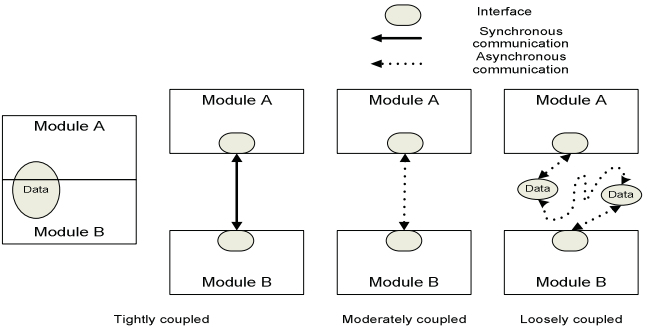


Fig. 1. Relationships between application modules. The last case shows a loosely coupled relationship between module *A* and module *B*

By such loosely coupled relationships, the influence among modules is much reduced. Partial failures won't cause the whole application to crash. Once execution resumption mechanisms are introduced, the application can complete its computation successfully even when normal errors or exceptions occur. Thus the robustness of the application can be guaranteed. The loosely coupled structure also reduces the difficulty of resource co-allocation since the resource allocation for one module affects little on the resource allocation for another module.

We define a *loosely coupled module (LCM)* as an individual module of a distributed application that can be scheduled independently onto remote or local resources. It can be either a remote service request or local processing. A LCM communicates with other parts of the application by loosely coupled messages. A *loosely coupled application (LCA)* is a distributed application that contains LCMs. How to schedule LCMs is an important part of work in the scheduling of the whole application. For convenience, in the rest of the paper, when we say scheduling of loosely coupled applications, we actually indicate scheduling of a group of LCMs.

The loosely coupled application model is part of our framework of Service-based Heterogeneous Distributed computing (SHDC) [7]. In our framework, implementing such loosely coupled applications mainly involves three aspects of work. On the application level, each component can be designed independently, and communication among these components is completed by either file exchanging or explicit messages. On the system level, tools are needed to administrate various services, schedule applications and coordinate data communication. On the SHDC framework level, services in the Internet are organized into a P2P network and distributed applications are scheduled by the cooperation of different peers [8]. Moreover, powerful description methods are required to enable an application and the system to understand each other. The descriptions shall have the ability to describe the necessary semantic contents of applications so that they can be scheduled rightly by system-level tools.

2.2 The Loosely Coupled Application Model

We define a loosely coupled application as a set *M* containing *n* modules: M_0, M_1, …, M_{n-1}. Each module M_i is composed of 5 elements: {$attribute_i$, {$inputmsg_{pi}$, …,

$inputmsg_{qi}$}, {$outputmsg_{jk}$, …, $outputmsg_{jm}$}, {$term_{pi}$, … $term_{qi}$}, L_i}. $Attribute_i$ holds the attributes of the corresponding component such as its functionality, execution requirements. The *attribute* element provides the necessary information of a component so that it can be scheduled onto proper resources. The element $inputbuf_{pi}$ holds messages received from module M_p ($0 \le p \le n-1$) but M_i hasn't processed the messages by internal computation steps. The element $outputbuf_{ik}$ holds messages that M_i wants to send to module M_k ($0 \le k \le n-1$) but the messages haven't been delivered out. The *term* element is used to monitor input messages to ensure they conform to the requirement of the corresponding modules. When any exception occurs on the input messages, the corresponding events will be issued by the *term* element so that the exceptions can be dealt with. A key function of the *term* element is to solve the non-response problems in an asynchronous system. It can satisfy the acceptable period of time of receiving a message. When the deadline reaches and the message has still not arrived, an exception event can be issued. Another element is the logic element *L* which monitors the output messages. It is used to produce communication events to transfer the output messages to other modules. It is useful especially when implementing logic controls among a group of components: when the output satisfies some conditions the data transfer can be re-directed by element L_i. This loosely coupled application model is a state-based model. Each state of module M_i contains three sets: {$inputmsg_{pi}$ …$inputmsg_{qi}$}, {$outputmsg_{ik}$…$outputmsg_{im}$}, {$term_{pi}$ … $term_{qi}$}. The *attribute* element and the logic element are not included in M_i's state since they are predefined and unchangeable. The state set Q_i contains a distinguished subset of initial states and a distinguished subset of terminal states. In an initial state every $inputmsg_{pi}$ must be empty.

The module M_i's states, except for the $outputms_{gi}$ (because in an asynchronous system, the computation will be triggered by the input, the output will not influence the state transition), comprise the accessible states of M_i. When the transition function accepts an input value of the accessible state of M_i, it produces a value of the accessible state of M_i as output in which outdated data in $inputbuf_i$ is cleared. It also produces as output at most one incident message to every other module in M. Each step processes the necessary messages waiting to be delivered to M_i and results in a state change and at most one message to be sent to every other module. When there is an input message $inputmsg_{ij}$ or an output message $outputmsg_{pq}$, we say that module j is dependent on module i, or module q is dependent on module p, denoted by *DEP(i,j)* and *DEP(p,q)* respectively.

There are five kinds of normal events in the system. One kind is a computation event, denoted *computation(i)*, representing a computation step of module M_i in which M_i's transition function is applied to its current accessible state. When a component finishes its computation, a *finishcomputation(i)* event will be created denoting that the computation result is ready to be further used. Another kind of events is a delivery event, denoted *delivery(i,j,m)*, representing the delivery of message m from module M_i to M_j. The fourth kind of events is exception events, denoted *termexception(p,i,m)*, representing that module M_i cannot receive (or accept) message *m* from module M_p according to the setting of $term_{pi}$ due to either a network exception or a computation error or a service fault. The fifth kind of events is communication events, denoted *communication (i,j,m)*, representing that module M_i will send message *m* to module M_j.

A configuration is a vector

$$C = (q_0, \ldots, q_{n-1})$$

where q_i is a state of module M_i. The states of the *outputmsg* variables in a configuration represent the messages that are in transit on the communication channels. An initial configuration is a vector $(q_0, \ldots q_{n-1})$ such that each q_i is an initial state of M_i. The behavior of a system over time is modeled as an execution, which is a sequence of configurations alternating with events. An execution is a (finite or infinite) sequence of the following form:

$$C_0, \varphi_1, C_1, \varphi_2, C_2, \varphi_3, \ldots$$

where each C_k is a configuration and each φ_k is an event. If the execution is finite then it must end in a configuration. Furthermore, several conditions must be satisfied:

1. If φ_k = *delivery(i,j,m)*, then *m* must be an element of $outputmsg_{ij}$ in C_{k-1}. The only changes in going from C_{k-1} to C_k are that *m* is added to $inputmsg_{ji}$ in C_k. In another word, a message is delivered only if it is arrived and the only change is to copy the message from the sender's outgoing message buffer to the recipient's incoming message buffer.

2. If φ_k = *computation(i)*, then the only changes in going from C_{k-1} to C_k are that M_i changes state according to it's transition function operating on M_i's accessible states in C_{k-1} and M_i will not be accepting any input messages during the computation time to ensure the computation can be rightly implemented.

3. If φ_k =*finishcomputation(i)*, then the only changes in going from C_{k-1} to C_k are that M_i changes state according to its transition function and the output messages are produced and ready for further communication. At the same time, the set of input messages specified by M_i's transition function are removed from $outputmsg_{pi}$ in C_k.

4. If φ_k = *termexception(i,p,m)*, then the only changes in going from C_{k-1} to C_k are that either a communication request is sent out to re-transfer messages *m* from module M_p (when the *termexception* event is caused by a network failure) or the state of M_p is set to it's initial state (when the *termexception* event is caused by a computation failure), and then reset the term settings relevant to module M_p in C_k.

5. If φ_k = *communication(i,j,m)*, the system doesn't change its state. Moreover, the message *m* is supposed to be still accessible after the communication events and delivery events occur to ensure the application's robustness. After the asynchronous message *m* has been sent from M_i to M_j, a delivery event will be produced.

We assume that all events are produced by a unified application controller or can be notified to the controller before the next actions. Thus all of the events in the model can be arranged into an event queue by a unified time. We use *time(q)* to denote the time that event q occurs.

The execution time of a module M_i at phase *k* is:

$$execT(i_{phaes\text{-}k}) = time(finishcomputation(i_{phaes\text{-}k})) - time(computation(i_{phaes\text{-}k}))$$

The transfer time of a message *m* from module M_i to M_j at phase *k* is:

$$transT(i,j,m_{phaes\text{-}k}) = time(delivery(i,j,m_{phaes\text{-}k})) - time(communication(\ i,j,m_{phaes\text{-}k}))$$

The execution time of the application is:

$$T = max\ \{time(delivery\ (i,j,m))\}$$

$0<=i,j<=n-1$, m is any possible message that carries the results of the computation.

The unstable network status and heterogeneous Internet services are two important characteristics of Grid environments. The robustness of an application becomes a basic requirement. The loosely coupled application model proposed here emphasizes the fault-tolerant issue in two aspects. One of the functions of the *terms* in the model can be used to solve the problems of asynchronous messages passing. With these terms, network faults and service faults can be detected, the corresponding data can be re-delivered and necessary application modules can be re-scheduled. Communication between different applications modules are implemented by buffered asynchronous message passing. Each output buffer is reserved until the data in the buffer has been dealt with by the next step in the computation successfully. This buffer mechanism may lead to some storage waste, but can guarantee computations to be executed accurately even when a network fault or a service fault occurs.

The *termexception* event is the key to detect a network or resource exception. The *termexception* events are created by the constraints of input messages. Application designers can set term elements to ensure the input messages satisfy some conditions. When a message cannot satisfy a condition, a *termexception* event will be created. Usually a term element $term_i$ is in such a form:

$$term_i: \{message\ m;\ condition\ c;\ actions\}$$

It implies that when message m cannot satisfy condition c, the following actions will be issued. Usually those actions are to create some *termexception* events. The condition c can be any conditions to restrict the messages, such as message size, message precision, etc. Moreover, one important function of the term element is to limit the arrival times of messages. Actually this functions like a timer. When an appointed message has not arrived during the prescribed time period, the corresponding events will be created to inform the system or other modules.

Suffering from the explicit message communication, applications designed in this model might meet problems for some uncertain messages. For an example, if one component in the application is defined as:

```
component A
{
…
if (condition1) then send message m to component B;
if (condition2) then send message m to component C;
…
}
```

To deal with the uncertain messages, the logic element can be introduced to implement control logics at the component-level. Each logic element L_i is in such a format:

$$L_i: \{message\ m;\ condition\ c;\ actions\}$$

in which m is a message, c is a logic expression, *actions* are usually communication events. When message m satisfies the condition c, the *actions* will be issued.

To apply the logic element to the above example (suppose components *A*, *B* and *C* are included in modules M_i, M_j and M_k respectively):

```
component A
{
…
produces message m;
…
}
L0: {m; condition1; communication(i,j,m)}
L1: {m; condition2; communication(i,k,m)}
```

By producing communication events according to conditions at runtime, the loosely coupled application model can implement complex logic controls such as branches and loops at the component-level. In this way, the modularization of each component is largely enhanced. The complexity of the design of each component is reduced and the schedulability of the whole application is improved.

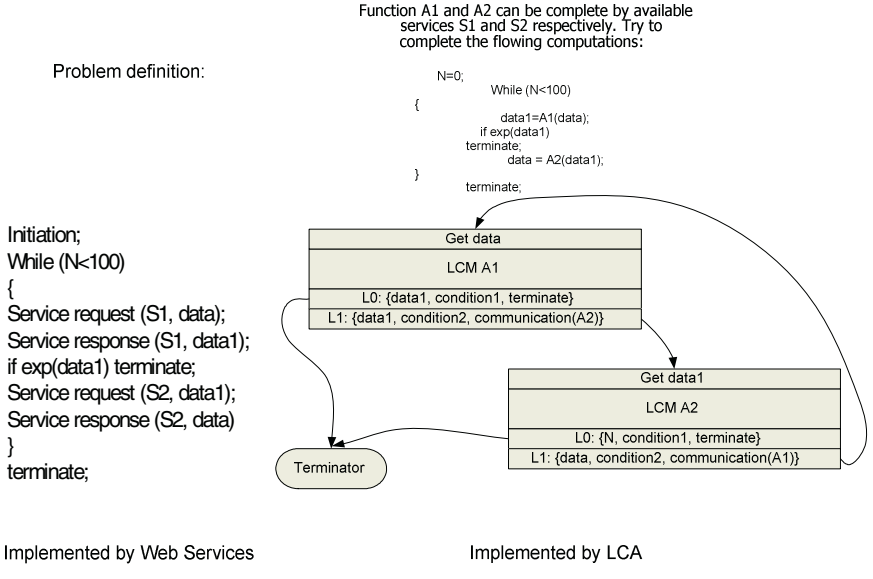


Fig. 2. An example application implemented by Web services and the loosely coupled model. The implementation by LCMG is more modularized than by Web services. It introduces only two modules and at most 200 data communications. The implementation of Web services style may cause 200 possible service requests and 400 possible data communications

This loosely coupled application model is based on services. Also there are other service-based models such as Web services and Grid services. The most important

difference between this model and other work is that modules of an application in our model are absolutely loosely coupled: each module only interacts with the outside world by reading or writing predefined format of messages; different modules do not need to know each other even when they are dependent; the global execution flow is understandable to the system so that various exceptions can be caught and dealt with; applications' robustness can be guaranteed. Distributed applications in the loosely coupled model are well modularized and each module in the applications is comparatively independent. By Web services or Grid services methods, services are called from programs, and application developers must face various exceptions caused by the network or services. But in our model, each module in an application can be designed independently, and can be developed in any programming languages, software or hardware tools and remote services. As long as the interfaces of the modules are correctly designed, the application can be scheduled by the Grid management system efficiently. Moreover, data caching is an important feature of the model. Input and output can be cached on the server side, and they can be transferred to any other server by the direction of the application scheduler dynamically. By using proper scheduling algorithms, the communication cost can be dramatically reduced, and at the same time, applications can be executed with better performance. We give an example application that implemented by Web services and LCMG respectively in Figure 2.

3 Schedule of Loosely Coupled Applications

The procedure of mapping an application onto computing resources according to some rules to implement the computation is called task scheduling. The objective of task scheduling is to order the execution of applications so that task precedence requirements are satisfied and a minimum schedule length is provided. Task scheduling is one of the most important subjects that have been extensively studied in parallel computing and distributed computing. The loosely coupled application model we proposed in our framework largely weakens the relationship between different application components, and provides an explicit structure to increase the schedulability of Grid applications. The efficiency and effectiveness of scheduling algorithms can largely influence the performance of the application. There are many scheduling algorithms based on various computing platforms. But traditional scheduling algorithms are mostly based on shared-memory systems or a cluster of workstations, they cannot be used on such heterogeneous scheduling problems. In this section, we present the definition of scheduling a loosely coupled application onto a heterogeneous distributed system.

We define a heterogeneous distributed system D as: $D = \{S, C, T, P\}$, where T is the attribute set of services; S is a finite set of services, each element S_i represents a service that can be employed by applications, for $\forall\ S_i \in S$, $S_i = \{T_i \mid T_i \subseteq T\}$; C is a communication cost matrix, for $\forall\ 1 <= i, j <= N$, $C_{ij} \in C$, C_{ij} is the communication cost between service S_i and S_j, and P is a set of dependency functions describing the dependence relationship between different services, for $\forall\ P_i \in P$, $P_i = f_i(S_{r0}, \ldots S_{rk})$, denoting that service S_i is dependent on services S_{r0}, S_{r1}, … and S_{rk}, and there is a function $f_i()$ that can be used to compute the influence on performance that services S_{r0} to S_{rk} have on S_i.

A loosely coupled distributed application DA is defined as: DA = {MODULE, DATA, T, COST}, where MODULE is a finite set of n modules, DATA is a finite set of data, T is a finite set of attributes, COST is a finite set of functions to predict the performance of modules. For $\forall$ 0<j<n+1, $MODULE_j \in$ MODULE, $MODULE_j$ = {($input_j$, $output_j$, $cost_j$, AT_j)| $input_j \subseteq$ DATA, $output_j \subseteq$ DATA, $cost_j \in$ COST, $AT_j \subseteq$ T}. In this definition, $input_j$ is the set of data module $MODULE_j$ requires. Once $input_j$ is ready, the module $MODULE_j$ can start its execution. While $output_j$ is the set of data the module $MODULE_j$ produces. The function of $cost_j$ is used to approximately evaluate the computation cost of $MODULE_j$. AT_j contains attributes of the module.

A general schedule scheme is a map from the task graph to the target system: f: DA $\rightarrow$ D $\times$ [0, ∞]. f(i) = (S_j ,t_i) means module $MODULE_i$ is scheduled onto service S_j, and its predictable start time is t_i. A module $MODULE_i$ can be scheduled onto service S_j if $AT_i \subseteq Ts_j$. The time that all modules complete execution and return results is called the Schedule Length (SL). One of the aims of a scheduling algorithm is to reduce SL to as small a value as possible. Such a task scheduling problem is shown to be NP-complete [9]. As a Grid environment may contain a huge number of resources or services, it is impossible for the scheduling algorithm to map a loosely coupled distributed application based on all possible resources that can satisfy the application's requests. How to select appropriate resources and make better scheduling schemes is an important issue for ensuring both client's and system's performance.

We can use the definitions of the loosely coupled application model to define some parameters such as adaptability and schedulability to implement various scheduling algorithms for Grid applications. For example, we give a simple definition of adaptability below. Suppose there are limited services in the environment, the number of the services is N. For any module M_i in an application that contains n modules (0<=i<=n), if there are K services that can satisfy the requirements of module M_i, we denote the adaptability of module M_i as $adaptability(M_i)$ = (K-1)/N. For those modules that can only be scheduled onto one resource, the adaptability is always 0. The parameter of adaptability can be used in scheduling algorithms for applications to achieve better performance or for the system to keep its usability. For example, we can give a simple insufficient resource first algorithm based on the adaptability of each module. This algorithm schedules modules with lower adaptability first to avoid potential resource conflicts in later computations.

```
while (there are un-scheduled modules)
{
  while (there are ready-for-scheduling modules)
  {
     select a module with lowest adaptability: Mi;
       if (there is available resources to schedule Mi)
      {
          schedule Mi;
          mark Mi as scheduled;
       }
    }
}
```

4 Conclusions

In this paper, we have introduced a loosely coupled application model which can be used to model Grid applications. Comparing to other models, this application model is powerful to model Grid applications more directly and efficiently; and at the same time it can guarantee the robustness, adaptability, and schedulability of Grid applications.

Acknowledgements

The work described in this paper was partially supported by the following grants: RGC Competitive Earmarked Research Grants (Project ID: 2150348, RGC Ref. No.: CUHK4187/03E ; Project ID: 2150414, RGC Ref. No.: CUHK4220/04E).

References

1. M. Baker, R. Buyya and D. Laforenza, “Grids and Grid Technologies for Wide-Area Distributed Computing”, *Software - Practice and Experience 32*(15): 1437-1466 (2002).
2. I. Foster, and C. Kesselman, (Eds.), *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 2004.
3. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid", *International Journal on Supercomputing Applications* 15(3):200-222, 2001.
4. I. Foster, C. Kesselman, J.M. Nick, and S. Tueche. “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”, *Open Grid Service Infrastructure WG*, Global Grid Forum, June 2002.
5. C. Lee and D. Talia, “Grid programming models: current tools, issues and directions”, in *Grid Computing – Making the Global Infrastructure A Reality*, F. Berman, G.C. Fox, and A.J.G. Hey, (Eds.), John Wiley, 2003.
6. D. Bader, *et al*., “The Role and Requirements of Grid Programming Models”, www-unix.Gridforum.org/mail_archive/ models-wg/pdf00002.pdf
7. F. Wu and K.W. Ng, “A Toolkit to Schedule Distributed Applications on Grids”, *Fourth International Network Conference*, pp. 11-18, UK, 2004.
8. F. Wu and K.W. Ng, “SHDC: A Framework to Schedule Loosely Coupled Applications on Service Networks”, *Grid and Cooperative Computing* - GCC 2004: Third International Conference, Wuhan, China, October 21-24, 2004.
9. R.L. Graham. "Bounds on multiprocessing anomalies." *SIAM Journal of Applied Mathematics* , 17(2): 416-429, 1969

A Locking Protocol for a Distributed Computing Environment

Jaechun No[1] and Hyoungwoo Park[2]

[1] Dept. of Computer Software,
College of Electronics and Information Engineering,
Sejong University, Seoul, Korea
[2] Supercomputing Center,
Korea Institute of Science and Technology Information,
Daejeon city, Korea

Abstract. The need for distributed file systems has been growing for decades to provide clients with efficient and scalable high-performance accesses to stored data. The clients physically share storage devices connected via a network like GigaEthernet or Fibre Channel and, on those clients, distributed file systems take responsibility for providing coordinated accesses and consistent views of shared data. In such a distributed computing environment, one of the major issues affecting in achieving substantial I/O performance and scalability is to build an efficient locking protocol. In this paper, we present a distributed locking protocol that enables multiple nodes to simultaneously write their data to distinct data portions of a file, while providing the consistent view of client cached data, and conclude with an evaluation of the performance of our locking protocol.

Keywords: locking protocol, distributed file system, distributed computing, MPI-IO.

1 Introduction

Distributed file systems have been developed for decades to provide clients with efficient and scalable high-performance accesses to stored data. The clients are physically connected to one or more servers via a network like GigaEthernet or Fibre Channel [1, 2, 3, 4, 5, 7, 10, 11, 12], and, on those clients, distributed file systems take responsibility for providing coordinated accesses to remotely stored data and for providing consistent views of client cached data. In such a distributed computing environment, one of major considerations affecting in achieving substantial I/O performance and scalability is to build an efficient locking protocol.

One of the general locking protocols for a distributed environment is to provide a token-based lock manager, as described in [1, 2, 3, 7]. The basic idea behind the token-based lock manager is that before a client performs file data or metadata operations, it requires a related lock from the lock server. If there is no

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1066–1075, 2005.

conflicting request to the lock, or the client is the only requester to the lock, the lock server then grants the lock to the client.

A locking protocol to support data consistency and cache coherency has a significant effect on generating high performance I/O. For example, large-scale scientific applications in physics, chemistry, biology, and other sciences generate huge amounts of data and utilize them for data analysis, visualization and so on. In order to achieve high-performance I/O, many such applications use parallel I/O methods where multiple client nodes simultaneously perform their I/O operations. MPI-IO is among those parallel I/O methods.

MPI-IO [8, 15] is specifically designed to enable the optimizations that are critical for high-performance parallel I/O. Examples of these optimizations include collective I/O, the ability to access noncontiguous data sets, and the ability to pass hints to the implementation about access patterns, file-striping parameters, and so on. In order to achieve high I/O performance using MPI-IO on top of distributed file systems, the file system must provide the ability to lock a file per data section to have multiple concurrent writers to a file.

However, many of the locking protocols integrated with distributed file systems are based on a coarse-grained method [1, 2, 4, 5] where only a single client at any given time is allowed to write its data to a file, while the other clients are waiting for the current node to finish its write operation even when the others would write to the different data portions of the same file. This drawback significantly degrades I/O performance in many scientific applications where supporting parallel write operations happens to be proved generating high I/O bandwidth [6, 8, 9, 13].

In this paper, we present a distributed locking protocol based on multiple reader/single writer semantics for a data portion to be accessed. In this scheme, a single lock is used to synchronize concurrent accesses to a data portion of a file. However, several nodes can simultaneously run on the district data sections in order to support data concurrency. We conclude our paper by discussing performance evaluation of our locking protocol.

2 Design Motivation

Our main objectives in developing a distributed locking protocol were to provide high-performance parallel I/O, to minimize the communication latency occurred during the lock negotiation steps, and to utilize local lock services as much as possible.

- **High-performance I/O.** We designed the distributed locking protocol capable of allowing multiple concurrent writers to the same file to achieve high performance I/O. Also, the locking protocol provides data consistency between the data stored in the storage device and the data stored in the client-side cache. On top of the distributed file system integrated with this locking protocol, many data-intensive applications can generate high I/O bandwidth using parallel I/O libraries, such as MPI-IO, the I/O interface

defined as part of the MPI-2 standard [8, 15] as the standard, portable API for I/O in parallel applications.

- **Low communication latency.** We designed the locking protocol to reduce the network overhead taking place during the lock negotiation steps with Global Lock Manager (GLM). All the lock requests coming from the client nodes are evenly distributed on multiple GLMs. Moreover, in order to minimize the number of callback messages necessary to revoke and release a lock, we grouped all the client nodes into several node groups. If GLM finds the node group where the lock holder belongs to it then sends a lock revocation message to the node group. After the lock holder completes the corresponding callback function to release the requested lock, it sends back an acknowledge to GLM to grant the lock to the requesting client node.
- **Use of local lock service.** We designed the locking protocol to utilize local lock service as much as possible in order not to incur communication overhead with GLM. In order to use the local lock service to the maximum extents, we designed the distributed lock scheme using the lazy-revocation or sticky lock method [2] to retain privileges on data sections even in the absence of active processes on a client node. If process on a node accesses to the smaller data section than the section controlled by an already acquired lock and if the requesting lock mode does not conflict with the mode of the acquired lock, the lock held is then split and the lock of the requesting data section, called childlock, is returned to the process. However, the lock information of a childlock needs not be stored in GLM.

3 Implementation Details

3.1 Overview

Figure 1 illustrates the distributed lock interface that is integrated with distributed file systems. Applications issue I/O requests using local file system interface, on top of VFS layer. Before performing an I/O request, each client should acquire an appropriate distributed lock from GLM in order to maintain data consistency between the cached data on clients and the remote, shared data on servers. The lock request is initiated by calling the lock interface, *snq_clm_lock*.

As mentioned in section 2, in order to reduce the communication latency occurring at the lock acquire step, we grouped the client nodes into several node groups. In the current implementation, an eight bit integer is used to denote node groups. When a client acquires an appropriate lock to perform I/O operation, the bit corresponding to the node group where the client belongs to is set to 1. Also, if a client requests a lock to GLM, GLM first locates the node group where the lock holder belongs to and then sends a callback message to the nodes of the node group. When the lock holder receives the callback message, it releases the requested lock and sends back an acknowledge to GLM to grant the lock to the requester.

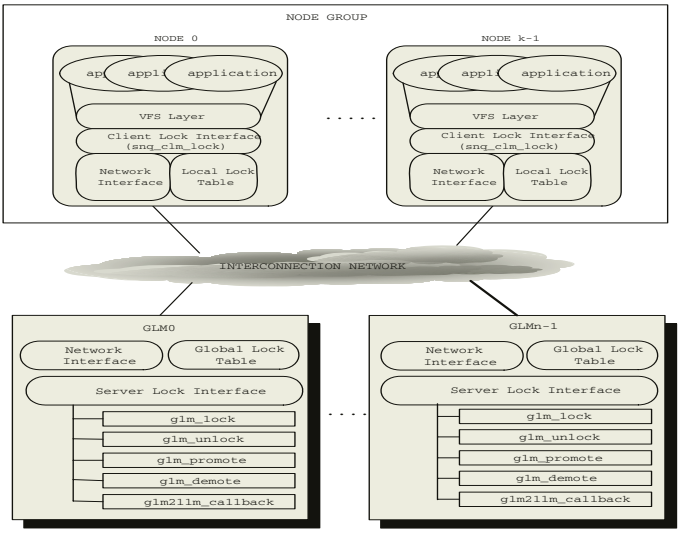


Fig. 1. A distributed lock interface

Figure 2 represents a hierarchical overview of the locking construct with two client nodes and one GLM. The lock modes that we provide for are SHARED for multiple read processes and EXCLUSIVE for a single write process. The lock structure consists of three levels: metalock, datalock, and childlock. The metalocks, inode0 on node A and inode1 on node B in Figure 2, synchronize accesses

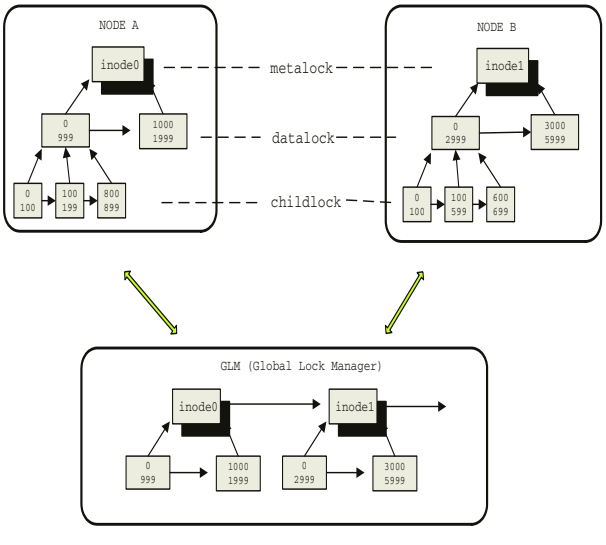


Fig. 2. A hierarchical overview of distributed locking protocol

to files and the value of a metalock is an inode number of the corresponding file. Below the metalock is a datalock responsible for coordinating accesses to a data portion. For example, on node A, metalock inode0 is split into two datalocks associated with the data sections 0-999 and 1000-1999 in bytes and, on node B, two datalocks below inode1 are associated with the data sections 0-2999 and 3000-5999 in bytes. In order to grant a datalock, the lock mode of the higher lock (metalock) must be SHARED, meaning that a file is shared between multiple clients.

The lowest level is a childlock that is of a split datalock. As mentioned in section 2, given that a datalock is granted, the datalock can be split further to maximize local lock services as long as the data section to be accessed by a requesting process does not exceed the data section of the datalock held. In other words, in Figure 2, the datalock for the data portion 0-999 is split into three childlocks that control accesses to the data portions 0-100, 100-199, and 800-899, respectively. The childlock is locally granted and therefore the requesting process needs not communicate with GLM to obtain the childlock. However, the childlock is granted only when the lock mode of a childlock is compatible with that of the higher datalock. The datalock and childlock are found by comparing the starting file offset and data length being passed from the local file interface.

GLM contains the global lock information consisting of a list of the locks that each GLM is responsible for serving. In Figure 2, GLM contains the metalocks, inode0 and inode1, and the datalocks of the data portions 0-999, 1000-1999, 0-2999, and 3000-5999 held by node A and node B. GLM also contains the node group information indicating those groups where the lock holders belong to.

3.2 Function Calls

Figure 3 represents the functions to be called to serve the lock request, lock release, and lock grant operations. The lock request operation is started by calling *snq_clm_lock* in the local file interface to read or write data. Once process finishes its I/O operation, *snq_clm_unlock* is called to wake up a sleeping process, if any, blocked while waiting for the lock to be released.

If the requesting lock exists in the local lock table or the data portion to be accessed does not exceed the data portion of an already acquired lock then the request is immediately satisfied and the lock is returned to process. If the requesting lock does not exist in the local lock table, indicating that either it is already held by a different node or the lock is requested at the first time, then *llm2glm_lock* is called to communicate with GLM. If the requesting lock mode is EXCLUSIVE, then *llm2glm_promote* is called to upgrade the lock mode.

GLM receives the lock service request and then calls *glm_lock* or *glm_promote* to grant a lock or to upgrade lock mode. *Glm_lock* and *glm_promote* both call a callback invoke function, *glm2llm_callback*, to send an appropriate callback message to remote clients. *Glm2llm_callback* invokes *send_callback_msg* that sends a message to the node group where the lock holder belongs to. After invoking *send_callback_msg*, *glm2llm_callback* is blocked until it is woken up by *glm_unlock* or by *glm_demote*. *Glm_unlock* is a function to be called to update the global

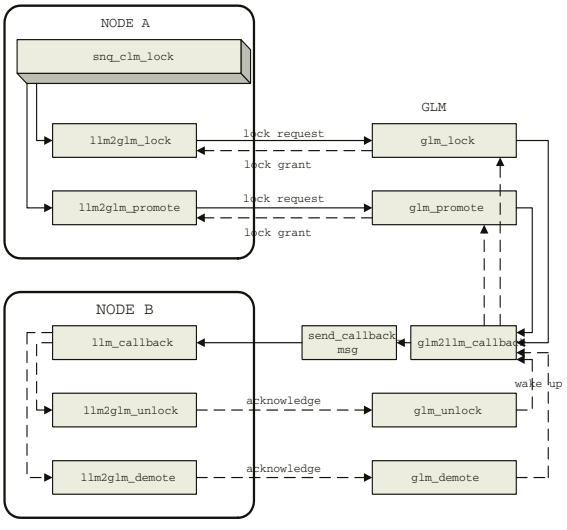


Fig. 3. Steps to acquire a distributed lock

information of the lock that has been released on a remote lock holder and *glm_demote* is of the lock that has been downgraded on a remote lock holder.

On a client node, once a callback message is received, the lock interface calls *llm_callback* to release or to downgrade the lock requested. The lock release operation is performed by calling *llm2glm_unlock* and the lock downgrade operation is performed by calling *llm2glm_demote*. After completing its intended operation, each function sends back an acknowledge to GLM to grant the lock to the requesting node.

4 Performance Evaluation

We measured the performance of the distributed locking protocol on the machines that have Pentium3 866MHz CPU, 256 MB of RAM, and 100Mbps of Fast Ethernet. The operating system installed on those machines was RedHat 9.0 with Linux kernel 2.4.20-8. On top of Linux kernel, we installed SANique cluster file system[14] which is a Linux-based software solution in a SAN environment. The performance results focused on the time to obtain locks by performing lock revoke, downgrade, and upgrade operations. The time to invalidate client cached data and to write dirty data to disk was not included in the evaluation.

Figures 4 and 5 represent the time to obtain the locks with the exclusive mode in write operations and with the shared mode in read operations, as the number of clients increases from 4 to 16. Also, in Figure 4, one machine was configured as a GLM and, in Figure 5, four machines were configured as GLMs. When four machines were configured as GLMs, each lock request is given to a

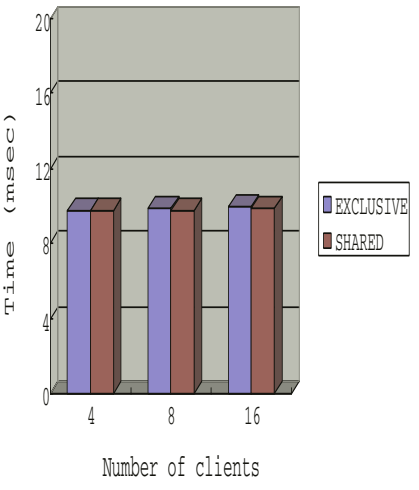


Fig. 4. Time overhead to acquire a distributed lock using one GLM. Each client read or wrote 1Mbytes of data to the distinct section of the same file

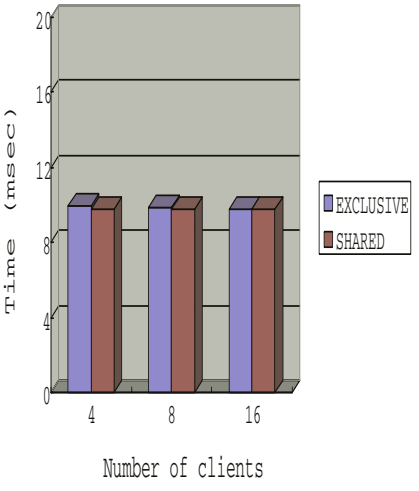


Fig. 5. Time overhead to acquire a distributed lock using four GLMs. Each client read or wrote 1Mbytes of data to the distinct section of the same file

GLM, according to round robin fashion. All clients read or wrote 1Mbyte of data to the distinct portions of the same file. In this case, the lock requested by each client is newly created on GLM and returned to the requesting client, causing no callback message to be sent to the remote lock holder.

Figures 6 and 7 show the time to obtain the locks with the exclusive mode and with the shared mode, while moving each client's data section to access to the one given to the neighbor at the previous step. Figures 6 and 7 both illustrate that the overhead of the lock revocation is significant with the exclusive mode because only a single client is allowed to write to a data section at any given time. With the shared mode, there is no need to contact the remote lock holder since a single lock can be shared between multiple nodes. With the shared lock mode, GLM just increases a counter denoting the number of shared lock holders before granting the lock.

Figures 8 and 9 show the elapse time to acquire locks, while changing the number of clients running on each node from 1 to 4. The total number of clients on every nodes was 16. As did in Figures 6 and 7, we changed the data access range of each client to the one given to the neighbor at the previous step. In this experiment, with two clients running on the same node, the callback message is sent to the remote lock holder every two I/O operations to revoke a lock. With four clients, the callback message is sent the remote lock holder every four I/O operations, resulting in the lock negotiation overhead decrement, compared to two clients on each node. According to this experiment, we could see that the dominating performance factor with 16 clients is the network overhead to contact the remote lock holder.

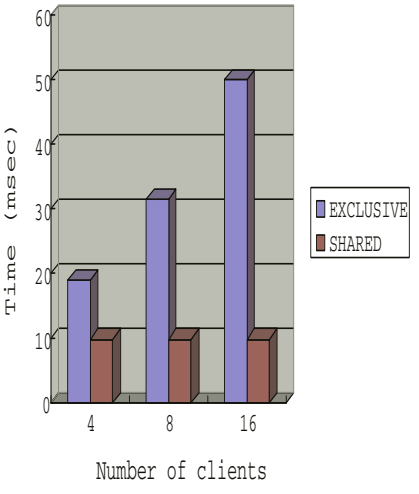


Fig. 6. Time to acquire a distributed lock using one GLM. A client's data section is shifted to the one given to the neighbor at the previous step

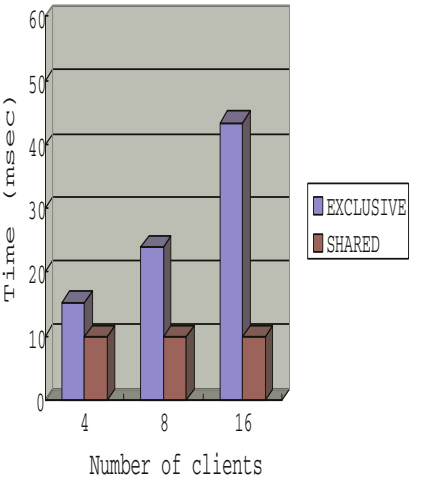


Fig. 7. Time to acquire a distributed lock using four GLMs. A client's data section is shifted to the one given to the neighbor at the previous step

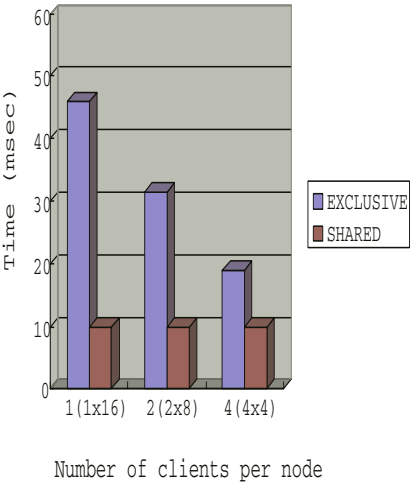


Fig. 8. Time overhead to acquire a distributed lock using one GLM as a function of number of clients running on each node. A client's data access range is shifted right at each step

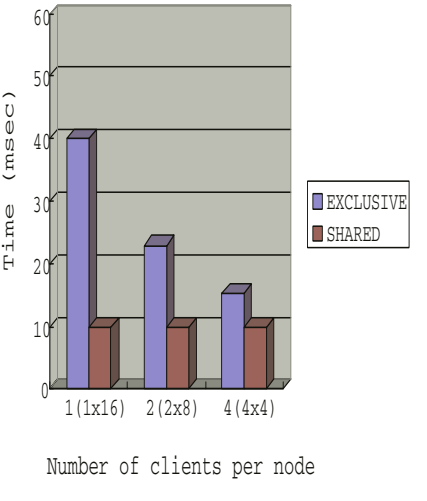


Fig. 9. Time overhead to acquire a distributed lock using four GLMs as a function of number of clients running on each node. A client's data access range is shifted right at each step

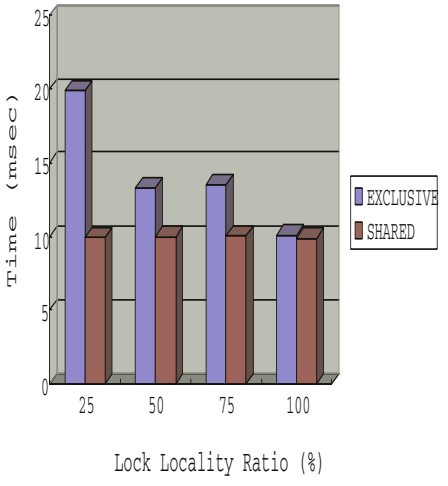


Fig. 10. Time to obtain a distributed lock as a function of lock locality ratio, using four clients with four GLMs

Figure 10 shows the effect of childlocks exploiting locality in the lock requests. The lock locality ratio means how often childlocks are taken; if lock locality ratio is of 25%, then 25% of total datalocks are childlocks needed to access to the smaller data portion than that of an already acquired datalock. If the lock locality ratio is of 100%, then all the locks are childlocks that do not incur communication overhead with GLM and remote lock holders. Figure 10 shows that, with the exclusive lock mode, the more childlocks are generated, the smaller time is taken to serve a lock request due to the drop in time to negotiate with GLM and remote lock holders. With the shared lock mode, however, the time to take a lock flattens out at about 9 msec because the remote shared lock holders need not give up the lock requested, allowing to have multiple lock holders with the shared mode. We believe that, however, more performance measurements must be conducted to verify the effect of lock locality.

5 Conclusion

Concurrent accesses to the same file frequently occur in a distributed computing where allowing parallel write operations significantly improves I/O bandwidth. However, most distributed client-server file systems support a coarse-grained locking protocol in which all the concurrent write operations to a file are serialized even when the data sections being written are different between writers. In this paper, we presented a distributed locking protocol with which several nodes can simultaneously write to the distinct data portions of a file, while guaranteeing a consistent view of client cached data. The distributed locking protocol has also been designed to exploit locality of lock requests to minimize

communication overhead with GLM and remote lock holders. As a future work, we plan to integrate the locking scheme with a SAN-based cluster file system, called SANique, developed by MacroImpact company.

References

1. Murthy Devarakonda, Bill Kish, and Ajay Mohindra. Recovery in the Calypso file system. ACM Transactions on Computer Systems, 14(3):287–310, August 1996
2. Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A Scalable Distributed File System. In Proceedings of the Symposium on Operating Systems Principles, 1997, pages 224–237
3. Edward K. Lee and Chandramohan A. Thekkath. Petal: Distributed Virtual Disks. In Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, 1996, pages 84–92
4. Kenneth W. Preslan, Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steven R. Soltis, David C. Teigland, and Matthew T. O'Keefe. A 64-bit Shared Disk File System for Linux. In Proceedings of Sixteenth IEEE Mass Storage Systems Symposium Seventh NASA Goddard Conference on Mass Storage Systems & Technologies, March 15-18, 1999
5. Steven R. Soltis and Thomas M. Ruwart and Matthew T. O'Keefe. The Global File System. In Proceedings of the Fifth NASA Goddard Conference on Mass Storage Systems, 1996
6. Jean-Pierre Prost, Richard Treumann, Richard Hedges, Bin Jia, Alice Koniges. MPI-IO/GPFS, an Optimized Implementation of MPI-IO on top of GPFS. In Proceedings of Supercomputing, November 2001
7. F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In Proceedings of the First Conference on File and Storage Technologies(FAST), pages 231–244, Jan. 2002
8. Rajeev Thakur and William Gropp. Improving the Performance of Collective Operations in MPICH. In Proceedings of the 10th European PVM/MPI Users' Group Conference (Euro PVM/MPI 2003), September 2003
9. Rajeev Thakur, William Gropp, and Ewing Lusk. Optimizing Noncontiguous Accesses in MPI-IO. Parallel Computing, (28)1:83-105, January 2002
10. Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A Parallel file system for Linux clusters. In Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000
11. K. Amiri, D. Petrou, G. R. Ganger, and G. A. Gibson. Dynamic function placement for data-intensive cluster computing. In Proceedings of the 2000 USENIX Annual Technical Conference, 2000
12. P. J. Braam. The Lustre stroage architecture. Technical Report available at - http://www.lustre.org, Lustre, 2002
13. Jaechun No, Rajeev Thakur, and Alok Choudhary. High-Performance Scientific Data Management System. Journal of Parallel and Distributed Computing, (64)4:434-447, April 2003
14. MacroImpact Inc., SANique CFS. A SAN Based Cluster File System, Version 2.1, Technical Report, August 2002
15. William Gropp and Ewing Lusk and Rajeev Thakur. Using MPI-2: A dvanced Features of the Message-Passing Interface, MIT Press, 1999, Cambridge, MA

Grid-Based SLA Management

James Padgett, Karim Djemame, and Peter Dew

Informatics Institute, School of Computing, University of Leeds,
Leeds LS2 9JT, United Kingdom
{jamesp, karim}@comp.leeds.ac.uk

Abstract. This paper presents an architecture for specifying, monitoring and validating Service Level Agreements (SLA) for use in Grid environments. SLAs are an essential component in building Grid systems where commitments and assurances are specified, implemented and monitored. Targeting compute resources, an SLA manager reserves resources for user applications requiring resources on demand. Methods for automated monitoring and violation capture are discussed showing how Service Level Objectives (SLO) can be validated. A SLA for a compute service is specified and experiments carried out on the White Rose Grid. Results are presented in the form of a SLA document and show the violations that were captured during task execution.

1 Introduction

Grids [1] offer scientists and engineering communities high performance computational resources supporting virtual organisations. In Grid environments, users and resource providers often belong to multiple management domains. Users need commitments and assurances on top of the allocated resources (this is sometimes referred to as Quality of Service), and it is the resource providers responsibility to deal with erroneous conditions, fail over policies etc. A key goal of Grid computing is to deliver the commitments and assurances on top of the allocated resources which include for example availability of resources (compute resources, storage etc), security and network performance (latency, throughput) [2].

Commitments and assurances are implemented through the use of Service Level Agreements (SLA), which determine a *contract* between user and Grid Service provider. A SLA is defined as an explicit statement of the expectations and obligations that exist in a business relationship between the user and the Grid Service provider. A formalised representation of commitments in the form of a SLA document is required, if information collection and SLA evaluation are to be automated. At any given point in time many SLAs may exist, and each SLA in turn may have numerous objectives to be fulfilled.

In the context of a Grid application, consolidation of management information is required when resources are spread across geographically distributed domains. SLAs may be distributed, and their validation depends on local measurements. With this is mind, the paper presents an SLA management architecture with automated SLA negotiation, monitoring and policing mechanisms. The current Open Grid Services Architecture (OGSA) [3] specification defines SLA management as a high level service

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1076 – 1085, 2005.

supporting SLAs within the Grid. Thus, a Grid user accessing Grid services on demand and with quality of service (QoS) agreements enabling commitments to be fulfilled is a primary requirement. The user can specify a SLA which will guarantee resources, provide job monitoring and record violations if they are detected. The SLA document records agreement provenance allowing for auditing mechanisms after it has terminated.

The structure of the paper is as follows: in section 2, the Grid SLA management architecture is presented. In section 3 an SLA specification for a compute service is outlined. Section 4 details the interaction between the SLA manager and a resource broker. In section 5 the monitoring engine and violation capture mechanism are demontrated. Section 6 presents some experiments involving a performance evaluation of the SLA Manager, and discusses the experimental results obtained on the White Rose Grid [4]. Related work is described in section 7, followed by a conclusion and discussion on future work.

2 Service Level Agreement Management Architecture

The SLA Management Architecture (Figure 1) defines an SLA Manager that runs in parallel with Grid Services to provide SLA management in environments such as the DAME Diagnostic Portal [5]. A client uses portal access to invoke Grid services which have time or performance requirements for their execution.

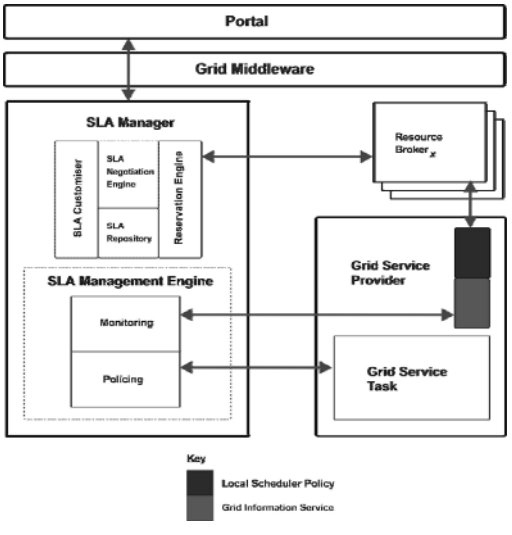


Fig. 1. SLA Management Architecture

2.1 SLA Management Interaction

Once the SLA Manager has instantiated an agreement and the Grid Service task is executing, interactions between the SLA Manager and the Grid Service task are maintained throughout the agreement life cycle to enforce the SLA guarantees.

2.2 Automated SLA Negotiation

The SLA Manager provides SLA negotiation using the Service Negotiation and Acquisition Protocol (SNAP) [6]. It provides a protocol for the negotiation and reservation of resources in order to guarantee the SLA based on a set of task requirements. These requirements are formally captured in a number of Service Level Objectives (SLO). The type of reservations made can be resource based or service based and will be executed through the Reservation Engine. The first iteration of which will implement performance-based Service Level Objectives. Further development will see the functionality enhanced. Once an agreement has been reached the Service Level Agreement is formalised into an SLA document. The SLA is offered to all parties for signing before being stored in the SLA repository where it can be accessed whenever validation is needed. The SLA Customiser has the ability to change the SLA document after the agreement has been signed; for changes in state or recording violations.

2.3 Automated SLA Monitoring

Once a SLA has been agreed and the resources have been reserved, the Grid task can begin execution. The SLA management engine is tasked with automated monitoring of the metrics needed to enforce the guarantees in the Service Level Objectives. It uses an external Grid Monitoring Service [7] to select the Grid monitoring tools which will be needed to monitor the SLA, such as Net Logger [8] and the Network Weather Service [9]. Tools such as these enable the SLA management engine to automatically monitor the SLOs based on dynamic resource information.

2.4 Automated SLA Policing

The SLA policing engine will adapt a task execution using a number of adaptation outcomes, examples of which include modifying the TTL (time to live), resource renegotiation, migration and termination. The adaptation feedback control mechanism will determine which of these outcomes is appropriate for a given situation. It will do this either in response to or to prevent violation of a SLO. The method used to adapt the Grid task is recorded in the SLA and based on the policies of the Grid resource. Adaptation has the potential to significantly improve the performance of applications bound with SLAs. An adaptive application can change its behaviour depending on available resources, optimising itself to its dynamic environment. For example, when resource load changes, a Grid system could seek to improve the quality of its compute resources or re-locate to another compute resource. To support adaptation feedback control, mechanisms for decision making are to be implemented using Fuzzy Control [10], however mathematical modelling, knowledge-based heuristics and reflection could be used.

3 SLA Specification

The SLA Document is XML based, whereas the SLA is represented internally by a content tree made up of Java objects. The SLA Manager supports a number of service

guarantees through differentiated classes of service; best effort, best effort with adaptation, reservation, reservation with adaptation.

3.1 Example: Specification for a Compute Service

An example SLA for a Compute service is specified in Table 1. It gives indication of the components which make up the SLA generated by the SLA Manager.

Table 1. SLA Specification of a compute service

Component	*Observation*
Purpose	A Grid job guaranteeing task requirements
Parties	Consumer, resource broker & compute resource
Scope	Compute service
Service Level Objective (SLO)	Ensure availability of resources satisfying task requirements for the duration of the Grid service task
SLO Attributes	CPU count, type, speed, usage. OS and OS version
Service Level Indicators (SLI)	For each SLO attribute, its value is a SLI
Exclusions	Adaptation / reservation may not be included
Administration	SLO's met through resource brokering / adaptation

The SLO's represent a qualitative guarantee such as CPU, RAM or HDD SLA. They comprise a set of SLI parameters which represent quantitative data describing the level of the SLO guarantee, such as CPU_COUNT or RAM_COUNT. The SLI values may take a number of forms, two which will be used are (1) a parameter distribution where the value of the SLI must be in a range or (2) a list where the parameter must equal a definite value

3.2 SLA Negotiation

The task requirements are based on the natural language definition given in Table 1. They are represented internally as a SLA content tree based and governed by a schema document. The negotiation engine parses the SLA content tree into an unsigned SLA document. This is passed to a resource broker which attempts to match the resource requirements by gathering information about the available Grid resources. A reservation decision is made based on the permissions and the suitability of the resource given the requirements. The SLA is signed by the SLA manager on behalf of the provider after which it is offered to the user for signing.

4 Resource Reservation

The SLA manager automates the SLA life cycle from negotiation to termination. An SLA document is specified in XML. The heterogeneous nature of XML makes it suitable for the SLA document in a service oriented architecture allowing the document to traverse the Grid along with the task execution. The SLA manager can be executed

on all Grid resources as a high level Grid Service within any OGSA compliant middleware. It parses SLA documents and configures the SLA Management system for the task execution. Once parsed the SLA manager un-marshalls the SLA document into objects which represent the content and organisation of the SLA. The architecture allows an SLA document to be updated as the Grid execution is progressing, allowing for violations to be captured and recorded as they are detected.

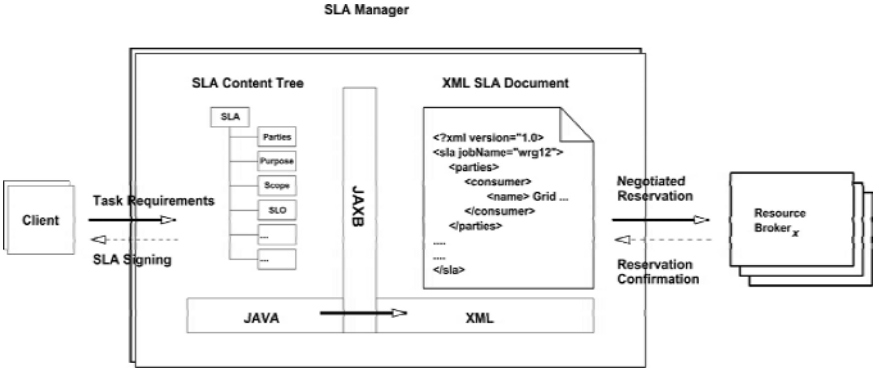


Fig. 2. Compute Service SLA Specification and Interation with Resource Broker

The SLA Manager's negotiation engine is able to interface with a resource broker providing reservations for different types of Grid resources as shown in Figure 2. It uses a resource broker to provision resources needed to fulfil the SLA. The example used in this work is the SNAP-based resource broker [11] which provides reservations for compute resources. This is similar to the Task Service Level Agreement (TSLA) defined within the SNAP framework which represents an agreement that specifies the desired performance of the task [6] where a task represents a job submission, execution or Grid service invocation. The SLA manager enters into an agreement with the resource broker service which provides a reservation guarantee, but it is the individual brokers which form the reservation agreements with the local Grid resources.

5 Automated Monitoring

5.1 Grid Measurement Tool

A Grid Information Service [12] (GIS) provides resource and service information from local Grid Monitoring Tools [13] that interface with low level Information Service Providers. The ability to use a number of low level Information Service Providers means that the Service can be used as a homogeneous interface to a number of Grid Monitoring Tools such as those mentioned in section 2.3. The use of a Grid Monitoring Tool (GMT) compliant with the Grid Monitoring Architecture (GMA) [7] allows a publish / subscribe mechanism to query resource information. To demonstrate the concept of automated monitoring of SLA's within a Grid - the Globus Monitoring and Discovery System (MDS) is used.

5.2 Monitoring Engine

The monitoring engine automates the SLA monitoring process. It matches SLO's to relevant Grid monitoring tools so that validation can be made against performance measurement data. A Grid integrates heterogeneous resources with varying quality and availability, this places importance on the ability to dynamically monitor the state of these resources and deploy reliable violation capture mechanisms.

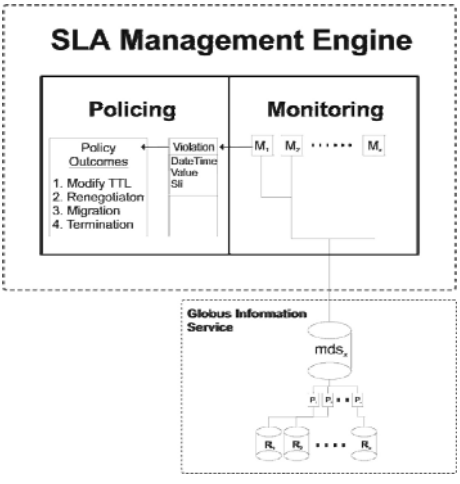


Fig. 3. Monitoring and Policing Mechanism

The monitoring engine launches a number of monitoring threads [M_x] corresponding to a SLI within each SLO (Figure 3). The threads periodically compare the measured value [m_x] taken from the Grid Information Service [12] [mds_x] with the value listed in the SLI, [sli_x]. When a violation is detected a violation object is created and passed to the policing engine.

5.3 Capturing Violations

The monitoringThreads compare the observed value [m_x] with that expressed in the SLA, [sli_x]. If a violation occurs a dateTime stamp is added to the SLO_x and the violated value of [sli_x] is recorded. The violation is represented as a violation object. It is passed to the policing engine and the SLA Customiser so that the violation can be recorded within the SLA document.

6 Experiments

6.1 Overview

The SLA Manager is expected to provide a managed Grid execution with guarantees over best effort execution through resource reservation, automated monitoring and

adaptation. Experiments are conducted as a proof of concept test of the SLA Manager in a production Grid environment; the White Rose Grid [4]. Once a SLA has been specified, the objectives are to: (1) Show the SLA manager can specify a SLA for a Grid compute service; (2) Test the effectiveness of the monitoring engine and the Grid monitoring tool; and (3) Determine if the violation capture mechanism is reliable and can keep pace with the monitoring engine.

The White Rose Grid was used, consisting of resources at the University of Leeds, Sheffield and York. The resources are configured with Solaris OS 5.8, Globus 3.0 middleware and MDS2.2 Grid information system. Communication occurs over a fast ATM network, called YHMAN. Network bandwidth and latency are not guaranteed within the SLA.

6.2 Experimental Design

Experimental Scenario

A user wants to execute a DAME Grid compute service task called, eXtract Tracked Orders (XTO) [14] with task requirements which they want formalising in a SLA. They would like guarantees on service performance to guarantee a time for completion of the task. The SLA Manager will take these requirements, specify a SLA, launch the Grid service, monitor its execution and record any violations. Two SLO's are specified: (1) the amount of RAM with an SLI of 3100MB; and (2) the free CPU_LOAD with a SLI of 0.25. The XTO Grid service is executed and monitored for a period of 130 seconds. Both SLO's are perturbed 80 seconds into the execution by two additional processes which are CPU and RAM intensive. The monitoring engine will launch two monitoring thread entries within the MDS: Mds-Memory-Ram-Total-freeMB and Mds-Cpu-Total-Free-5minX100.

6.3 Performance Results and Discussion

The experiments involve the submission and monitoring of the XTO Grid Service given a set of task requirements. Those requirements are to guarantee a minimum amount of free memory and CPU load on the compute resource. The monitoring threads sample MDS entries Mds-Memory-Ram-Total-freeMB and Mds-Cpu-Total-Free-5minX100 at each time interval from start to finish. Figure 4 shows the amount of free RAM and CPU_LOAD respectively during the task execution.

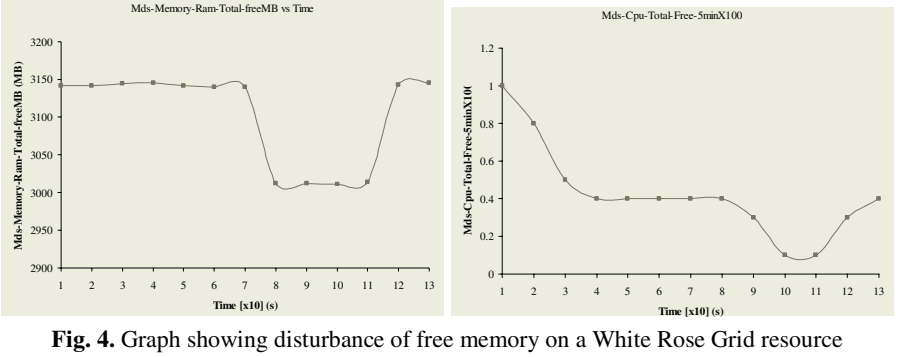


Fig. 4. Graph showing disturbance of free memory on a White Rose Grid resource

Figure 5 is the SLA after task completion. The monitoring engine successfully captures a timeStamp and value for each violation.

```
<ram>
    <count>3100</count>
    <violation slo="count">
       <timeStamp>2004-21-07T10:41:00Z</timeStamp>
       <value>3012</value>
    </violation>
    <violation slo="count">
       <timeStamp>2004-21-07T10:42:00Z</timeStamp>
       <value>3012</value>
    </violation>
    <violation slo="count">
       <timeStamp>2004-21-07T10:43:00Z</timeStamp>
       <value>3011</value>
    </violation>
    <violation slo="count">
       <timeStamp>2004-06-07T10:44:00Z</timeStamp>
       <value>3014</value>
    </violation>
</ram>
```

```
<cpu>
    <version>Sparc</version>
    <count>2</count>
    <speed>2000</speed>
    <free_load>0.25</free_load>
    <violation slo="free_load">
       <timeStamp>2004-21-07T10:42:00Z</timeStamp>
       <value>0.2</value>
    </violation>
    <violation slo="free_load">
       <timeStamp>2004-21-07T10:43:00Z</timeStamp>
       <value>0.2</value>
    </violation>
</cpu>
```

Fig. 5. SLA documents showing violations to RAM_COUNT and CPU_LOAD

7 Related Work

There have been a number of attempts at defining an SLA management architecture for both Web and Grid services. Architectures from Hewlett-Packard Laboratories [15] and IBM T.J. Watson Research Center [16] concentrate on service level agreements within commercial Grids. The service level agreement language used is that presented by Ludwig et al in [17]. The Global Grid Forum have defined WS-Agreement [18]; an agreement-based Grid service management specification designed to support Grid service management. Two other important works are automated SLA monitoring for Web services [19] and analysis of service level agreements for Web services [20]. Contract negotiation within distributed systems have been the subject of research where business-to-business (B2B) service guarantees are needed [21]. The mapping of natural language contracts into models suitable for contract automation [22] exist but has not been applied to Grid environments, nor has it been applied as a SLA. An approach for formally modelling e-Contracts [23] exists at a higher level than the research in [17]. Automated negotiation and authorisation systems for the Grid already exist [24] but involve no monitoring or policing ability post agreement.

8 Conclusion and Future Work

A formalised representation of commitments in the form of a SLA is required between users and service providers in order for QoS to be implemented in a Grid environment. SLA management is important in formalising QoS implementations within Grid services. Making simplifying assumptions regarding the task requirements, this work considers a SLA for a compute service, a mechanism for reserving resources and an automated monitoring / violation capture mechanism. The experiments were conducted in a Grid environment, the White Rose Grid. They show the SLA Manager is capable of negotiating and monitoring an SLA for a Grid compute service given a

set of task requirements. The monitoring engine / violation capture mechanism is effective at detecting and recording violations within the SLA.

The SLA Management architecture has provision for three types of SLA guarantees: performance, usage and reliability. The work presented here is targeting service performance; service usage and reliability will be the subject of further research.

Future work will centre on the implementation of a fuzzy controller for adaptation feedback control. It provides a methodology for implementing a more human heuristic control system by capturing qualitative control experience and applying it to the control algorithm. It is employed here in preference to a classic control method as they are inappropriate for Grid environments due to their heterogeneous nature. The use of a Fuzzy control technique is especially suited to this problem – where producing an accurate mathematic model for use in conventional control would be difficult.

Acknowledgements

The work reported in this paper was partly supported by the DAME project under UK Engineering and Physical Sciences Research Council Grant GR/R67668/01. We are also grateful for the support of the DAME partners, including the help of staff at Rolls-Royce, Data Systems & Solutions, Cybula, and the Universities of York, Sheffield and Oxford.

References

[1] Berman, F., *Grid computing : making the global infrastructure a reality*. Chichester: Wiley. 2003.

[2] Foster, I., C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 2001. **15**: p. 200-222.

[3] Foster, I., et al., *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, The Globus Project, June 22.

[4] *The White Rose Grid* [Online], White Rose Consortium, Available from World Wide Web: http://www.wrgrid.org.uk/

[5] Foster, I. and C. Kesselman, *The Grid 2 : blueprint for a new computing infrastructure*. Amsterdam ; Oxford: Morgan Kaufmann : Elsevier Science. 2004.

[6] Czajkowski, K., et al. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. IN: *Job scheduling strategies for parallel processing*. D.G. Feitelson, L. Rudolph, and U. Schwiegelshohn. 2002. Edinburgh: Berlin.

[7] Tierney, B., et al., *A Grid Monitoring Architecture*, Global Grid Forum, August 27.

[8] Gunter, D., et al. NetLogger: A Toolkit for Distributed System Performance Analysis. IN: *Modeling, analysis and simulation of computer and telecommunication systems*. 2000. San Francisco, CA: IEEE Computer Society.

[9] Wolski, R., N.T. Spring, and J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generations Computer Systems*, 1999. **15**(5-6): p. 757-768.

[10] Passino, K. and S. Yurkovich, *Fuzzy control*. Menlo Park, Calif. ; Harlow: Addison-Wesley. 1998.

[11] Haji, M., et al. A SNAP-based Community Resource Broker using a Three-Phase Commit Protocol. IN: *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium*. 2004. Santa Fe, USA.

[12] Schopf, J.M., A General Architecture for Scheduling on the Grid. *Journal of Parallel and Distributed Computing*, 2002(Special Issue on Grid Computing).

[13] Balaton, Z., et al., *Comparison of Representative Grid Monitoring Tools*, Laboratory of Parallel and Distributed Systems, Hungarian Academy of Sciences, Budapest, Hungary, March 2000.

[14] Nadeem, S., P. Dew, and K. Djemame, *XTO Grid Services on the White Rose Grid: Experiences in building an OGSA Grid Application*, DAME Technical Report, Informatics Institute, University of Leeds, UK,

[15] Sahai, A., et al. Specifying and Monitoring Guarentees in Commercial Grids through SLA. IN: *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*. 2003. Tokyo: IEEE Computer Society.

[16] Leff, A., J.T. Rayfield, and D.M. Dias, Service-Level Agreements and Commercial Grids. *IEEE Internet Computing*, 2003. **7**(4): p. 44-50.

[17] Ludwig, H., et al. A Service Level Agreement Language for Dynamic Electronic Services. IN: *Advanced issues of E-commerce and web-based information systems*. 2002. Newport Beach, CA: IEEE Computer Society.

[18] Andrieux, A., et al., *Web Services Agreement Specification (WS-Agreement)*, Global Grid Forum, July 26.

[19] Jin, L., V. Machiraju, and A. Sahai, *Analysis on Service Level Agreement of Web Services*, HPL-2002-180, HP Laboratories,

[20] Sahai, A., et al., *Automated SLA Monitoring for Web Services*, HPL-2002-191, HP Labs,

[21] Goodchild, A., C. Herring, and Z. Milodevic, *Business contracts for B2B*, Distributed Systems Technology Center (DSTC), Austrailia, 2000.

[22] Milosevic, Z. and R.G. Dromey. On Expressing and Monitoring Behaviour in Contracts. IN: *Enterprise distributed object computing*. 2002. Lausanne, Switzerland: IEEE Computer Society.

[23] [Marjanovic, O. and Z. Milosevic. Towards Formal Modeling of e-Contracts. IN: *Enterprise distributed object computing*. 2001. Seattle, WA: IEEE Computer Society.

[24] Lock, R. Automated contract negotiations for the grid. IN: *Postgraduate Research Conference in Electronics, Photonics, Communications & Networks, and Computing Science*. 2004. University of Hertfordshire, UK: EPSRC

A Heuristic Algorithm for Mapping Parallel Applications on Computational Grids

Panu Phinjaroenphan[1], Savitri Bevinakoppa[1], and Panlop Zeephongsekul[2]

[1] School of Computer Science and Information Technology,
[2] School of Mathematical and Geospatial Sciences,
RMIT University, GPO Box 2476V, Melbourne Australia
{pphinjar, savitri}@cs.rmit.edu.au, panlopz@rmit.edu.au

Abstract. The mapping problem has been studied extensively. However, algorithms which were designed to map a parallel application on a computational grid, such as MiniMax, FastMap and genetic algorithms have shortcomings. In this paper, a new algorithm, Quick-quality Map (QM), is presented. Experimental results show that QM performs better than the other algorithms. For instance, QM can map a 10000-task parallel application on a testbed of 2992 nodes in 6.35 seconds, and gives the lowest execution time whereas MiniMax and a genetic algorithm, respectively, take approximately 1700 and 660 seconds, but produce 1.34 and 6.60 times greater execution times than QM's.

1 Introduction

Computational grid has been introduced as a distributed computing paradigm that is able to interconnect heterogeneous networks and a large number of nodes regardless of their geographical locations [1]. This paradigm provides an access to tremendous computational power that can be harnessed for various applications. Parallel applications are developed to solve implementations of computational intensive engineering or scientific problems that require such power.

The main aim of solving such problems with a parallel application is to reduce the execution time. As a computational grid involves a large number of nodes, one of the challenging problems that needs to be addressed is to decide the destination nodes where the tasks of the application are to be executed. This process is formally known as the *mapping problem* [2].

Unfortunately, the mapping problem is known to be a non-deterministic polynomially bounded (NP) complete problem [3], which means that the problem is intractable and very time consuming. Hence, heuristic algorithms have been employed to solve the mapping problem. Two of these algorithms are MiniMax [4] and FastMap [5] which have the same scope as the heuristic algorithm adopted in this paper. Genetic algorithms (GAs) are another approach that can be applied to this problem. However, those algorithms have shortcomings, as these will be discussed in the next section.

In this paper, a new mapping algorithm, Quick-quality Map (QM), is presented. Experimental results from the evaluation of QM compared with MiniMax

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1086–1096, 2005.

[4] and a genetic algorithm show that QM performs better than the other algorithms. For example, when mapping a 10000-task parallel application on a testbed of 2992 nodes, QM gives the best solution. The mapping time of QM is 6.35 seconds whereas MiniMax and GA take about 1700 and 660 seconds, respectively.

2 Background and Related Work

In the literature, the mapping problem has been studied extensively. Researchers often focus on the specific models of parallel applications and parallel systems, and concentrate on optimising a particular metric. These three features then are used to differentiate between the studies of the mapping problem.

A parallel application is usually modelled by a graph. Task Interaction Graph (TIG) and Task Precedence Graph (TPG) – also known as Directed Acyclic Graph (DAG), are the traditionally tools used. A DAG is used to model a parallel application that the tasks have order of executions whereas a TIG a parallel application that the tasks are simultaneously executable [6]. In both models, there can be computational costs associated with the tasks (vertices), and communication costs with the communications (edges) between the tasks.

In this paper, the parallel systems are broadly categorised into *modern* and *legacy* systems. An example of a legacy system is the Massively Parallel Processors (MPP) node. Such node often consists of many processors. It is not uncommon to assume that users can assign a particular task to a particular processor. The network topology that links the processors together is static, such as torus and hypercube. Examples of modern systems are a cluster and a computational grid. Nodes and networks are two major resources. In general, users can only assign a task to a node, and the operating system takes care of which processor will execute the task if that node has more than one processor. The specific network topology is not assumed. A graph is also used to model a parallel system with computational costs associated with the processors/nodes (vertices), and communication costs with the links (edges) between the processors/nodes.

Two major optimised metrics are *communication* and *execution times* (or *costs*). The choice of which metric to optimise depends on the assumptions of the application and system models. The model of the application studied in this paper is a TIG and its associated costs are heterogeneous. The parallel system is a computational grid, and the costs associated with the nodes and networks are also heterogeneous. The metric to be optimised is the execution time of the application. In the literature, MiniMax [4] and FastMap [5] are the two algorithms which have the same scope as the process considered in this paper.

MiniMax is a suit of heuristic algorithms [4] consisting of three steps: *graph coarsening*, *initial partition* and *refinement*.

In the first step, the application graph is coarsened until the number of tasks falls below a predefined threshold. Coarsening is an approach of producing a new application graph with less number of tasks by merging a task with one of its neighbours to form a new task (see Fig.1 for an example).

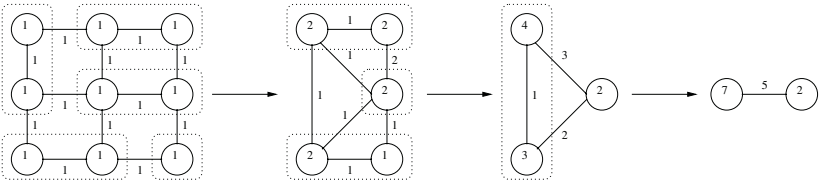


Fig. 1. An example of graph coarsening. After coarsening the application graph three times the number of tasks is reduced from 9 (finest) to 2 (coarsest)

In the second step, the coarsest application graph is mapped on the system with the *graph growing algorithm*. The algorithm maps high cost tasks to low cost nodes. The limitation is the algorithm can function only when the number of nodes is less than the number of tasks. Otherwise, a form of node selection is needed. This raises the issue of how to select the nodes since it is necessary to map tasks on nodes before judging whether the nodes should be chosen.

In the final step, the application graph is un-coarsened to produce the finer graph. During each un-coarsening to the finest graph, the execution time of the application is iteratively optimised with the *vertex migration algorithm*. The performance of the algorithm is the issue in this step as the complexity grows polynomially with both the numbers of tasks and nodes [7].

FastMap is also a suit of heuristic algorithms [5]. The optimisation is a genetic algorithm. A serious problem with FastMap is the assumption that all clusters have the same number of nodes. This is often not the case in real environments.

Genetic Algorithms (GAs) are a well-known optimisation technique. Braun et at. [8] studied the efficiencies of eleven algorithms by mapping independent tasks (zero on all communication costs) on heterogeneous parallel systems. A GA is among those algorithms and has shown to be one of the most efficient. Nevertheless, the high numbers of tasks and nodes can result in a massive search space. Thus, the computational cost of applying GAs could be prohibitively expensive, which significantly reduces their merit.

3 The Mapping Models

This section explains the models used to formulate the mapping problem.

3.1 The Parallel Application Model

An application is modelled as a weighted undirected graph $G = (V, E, W_V, W_E)$, where V is a finite set of vertices, and E a finite set of edges. An edge $e \in E$ is an unordered pair (v_x, v_y), where $v_x, v_y \in V$. V represents the tasks of application G, $|V|$ is the number of tasks, $W_V(v)$ the computational cost of task v, $e_{v_x v_y}$ represents the communication between tasks v_x and v_y (i.e. v_x and v_y

are neighbours), and $W_E(e_{v_x v_y})$ is the communication cost between tasks v_x and v_y. This model is the same as the ones used in MiniMax [4] and FastMap [5].

3.2 The Computational Grid Model

A computational grid is modelled by a *three-level-tree*. The levels are grid (g), cluster (c) and node (n) levels. Let $G' = (V', E', W'_V, W'_E)$ be a three-level-tree representing a computational grid. V' represents the nodes in G', $|C'|$ and $|V'|$ are the numbers of clusters and nodes in G', respectively, and $W'_V(v')$ is the computational cost of node v'. E' is a finite set of undirected edges. An edge $e' \in E'$ is an unordered pair $(v'_x, v'_y) \in V'$, $e_{v'_x v'_y}$ represents the communication between nodes v'_x and v'_y, and $W'_E(e_{v'_x v'_y})$ is the communication cost between nodes v'_x and v'_y.

A computational grid with a three-level-tree is specified according to the following rules. All nodes are in the same node level. When nodes can communicate to one another with the same communication cost, they can be grouped into the same cluster. A real cluster that has its nodes linked with the same network technology and medium is an example of a cluster in this model. An individual node is considered as a cluster of one node. All participating clusters (also the nodes) are in the same grid level and communicate through the grid network.

3.3 The Mapping Functions

When a parallel application, G, is mapped on a computational grid, G', the execution time of the application, $ET(G)$, is the execution time of the slowest node in G'; i.e.,

$$ET(G) = ET(v') \tag{1}$$

where $ET(v')$ is the execution time of the slowest node v'. $ET(v')$ is the sum of the computational and communication times of the tasks mapped on v'; i.e.,

$$ET(v') = \sum_{i=1}^{N} W_V(v_i) W'_V(v') + \sum_{i=1}^{N} \sum_{j=1}^{M_{v_i}} W_E(e_{v_i v_j}) W'_E(e'_{v' v'_j}) \tag{2}$$

where v_i is the i^{th} task mapped on node v', N the number of tasks mapped on node v', v_j the j^{th} neighbour of task v_i, M_{v_i} the number of neighbours of task v_i, and v'_j the node on which task v_j is mapped (see Fig.2 for an example).

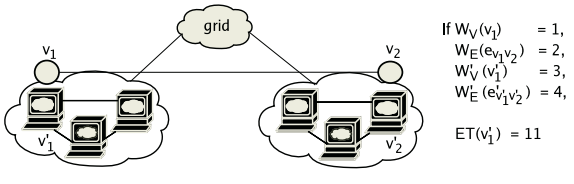


Fig. 2. A 2-task parallel application mapped on a grid modelled by a three-level-tree

4 QM Algorithm

Unlike the other algorithms that optimise the execution time of the slowest node, QM instead optimises the execution time of each task. The core idea is that each task is iteratively mapped on a new node such that the execution time of the task is lower than its current execution time. Given v as a task mapped on node v', the execution time of task v, $ET(v)$, is equal to the execution time of node v', $ET(v')$; that is,

$$ET(v) = ET(v') \tag{3}$$

The flows of QM algorithm are shown in Fig.3. The first step is to coarsen the application graph until the number of tasks is less than a threshold. Each task in the coarsest graph is mapped on a randomly chosen node, which becomes the *current node* of the task. The execution times of all chosen nodes are then calculated using (2). In this step, each task has its own execution time, which is equal to the execution time of its current node.

Iteratively, a better node for each task is searched. In the case that more than one such node exist, the task is mapped on the *best node*, which is the node that gives the task the lowest execution time. However, not all nodes in the environment need to be considered.

Let c' be a cluster in G', v'_x the node in cluster c' such that its execution time, $ET(v'_x)$, is lowest. If more than one such node exist, the node with the lowest computational cost, $W'_V(\cdot)$, is considered to be v'_x. Let V'_y be a set of nodes in cluster c' that their computational costs, $W'_V(V'_y)$, are less than the computational cost of v'_x, $W'_V(v'_x)$. Let v be a task to be mapped, and V'_z a set of nodes in cluster c', which the neighbours of task v are mapped on (see Fig. 4 for an example).

```
00. QM (G, G')
01.     while (|V| ≥ threshold)
02.         G = coarsen G;
03.     for (i = 0; i < |V|; i = i + 1)
04.         v'_r = randomly choose a node in G';
05.         map v_i on v'_r;
06.     update execution times of all nodes in G';
07.     do
08.         for (i = 0; i < ⌊max / log10 |V|+1⌋; i = i + 1)
09.             for (j = 0; j < |V|; j = j + 1)
10.                 v'_b = find the best node in G';
11.                 if (v'_b is found)
12.                     map v_j on v'_b;
13.                     update execution times of P(v_j), v'_b, N(v_j);
14.                     update v'_x and V'_y of relevant clusters;
15.     while (G = un-coarsen G is applicable)
```

Fig. 3. QM algorithm

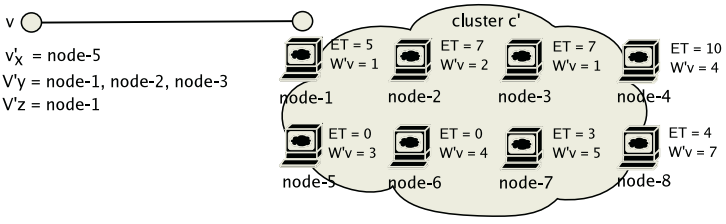


Fig. 4. Nodes in the cluster according to proposition 1

Proposition 1. *If the best node v'_b for task v exists in cluster c', then either $v'_b = v'_x$ or $v'_b \in (V'_y \cup V'_z)$ is true.*

Proof. Given the condition $v'_b \neq v'_x$ and $v'_b \notin (V'_y \cup V'_z)$ is true. Thus, before mapping task v, the execution times and computational costs of nodes v'_b and v'_x in (4) and (5) are true.

$$ET(v'_b) \geq ET(v'_x) \tag{4}$$

$$W'_V(v'_b) > W'_V(v'_x) \tag{5}$$

If the given condition is true, then the comparison of the execution time between nodes v'_b and v'_x after mapping task v in (6) must hold.

$$ET'(v'_b) < ET'(v'_x) \tag{6}$$

where ET' denotes the execution time after the mapping. (6) is equal to

$$ET(v'_b) + W_V(v)W'_V(v'_b) < ET(v'_x) + W_V(v)W'_V(v'_x) \tag{7}$$

However, (6) is false since $ET(v'_b) - ET(v'_x) \geq 0$ and $W'_V(v'_x) - W'_V(v'_b) < 0$. Hence, the given condition $v'_b \neq v'_x$ and $v'_b \notin (V'_y \cup V'_z)$ is false.

It can be seen that the number of nodes that each task needs to consider is reduced significantly without any impact on the mapping solution.

If the best node v'_b for task v_j is found, v_j is mapped on the best node, and the execution times of all relevant nodes are updated. These nodes are the previous node of v_j, $P(v_j)$, the best node, v'_b, and all the nodes on which the neighbours of task v_j are mapped, $N(v_j)$. v'_x and V'_y of the relevant clusters also need to be updated. The relevant clusters are the ones that the relevant nodes belong to. If the best node is not found, the task is remained on its current node.

Then, the application graph is un-coarsened, and the same optimisation is applied. This procedure is repeated until the application graph cannot be un-coarsened. The current nodes of the tasks are then the mapping solution.

Two parameters that have effect on the performance of the algorithm are *threshold* and the number of iterations for a task to find the best node. To avoid

the degrade in performance, the number of iterations is reversely proportional to the number of tasks (i.e. $\lfloor \frac{max}{\log_{10}|V|+1} \rfloor$). Preliminary experiments showed that $threshold = \sqrt{|V|}$ and $max = 13$ gave promising results.

5 Experiments

In the experiments, app-1, app-2 and app-3 consists of 100, 2500 and 10000 tasks representing small, medium and large scale parallel applications, respectively, are mapped on ten grid testbeds. The topology of the applications is the two-dimensional circular Cartesian. The number of neighbours of each task, and the computational and communication costs are randomly varied from 1 to 4. The testbeds are generated with different numbers of clusters and nodes (as shown in Table 1). In each cluster, the number of nodes is randomly varied from 1 to 64. The computational and communication costs are randomly varied from 1 to 10. All the costs in the application graphs and testbeds are the same as the ones used in MiniMax [4] and FastMap [5].

Table 1. The specifications of the grid testbeds

G'	grid-10	grid-20	grid-30	grid-40	grid-50	grid-60	grid-70	grid-80	grid-90	grid-100
$\|C'\|$	10	20	30	40	50	60	70	80	90	100
$\|V'\|$	359	615	914	1228	1679	1842	2535	2684	2915	2992

QM, MiniMax and a genetic algorithm (GA) are the experimented algorithms. It is preferable to compare FastMap with these algorithms. However, we are unable to do so due to incomplete information of the algorithm on how to calculate the execution time of the tasks during a mapping step.

There are two versions of QM: QM-1 and QM-2. QM-1 searches all nodes to find the best node, but does not coarsen the application graph. QM-2 searches the nodes according to proposition 1, and coarsens the application graph.

Since MiniMax cannot function if the number of tasks is less than the number of nodes (i.e. $|V| < |V'|$), in such situation, $|V|$ nodes are selected from the testbed. Three selecting algorithms are employed, and hence there are four versions of MiniMax: MiniMax-1, MiniMax-2, MiniMax-3 and MiniMax-4.

MiniMax-1 selects $|V|$ nodes as to minimise their total communication cost. This selection algorithm is used to select the nodes for Cactus (an astrophysics application) [7]. MiniMax-2 selects $|V|$ nodes that have the lowest computational costs while MiniMax-3 randomly selects $|V|$ nodes. MiniMax-4 functions in the situation that $|V| \geq |V'|$, and no node selection is required. The GA implemented is the same as the one used in [8].

The experiments are conducted on a 2.8 GHz Pentium-4 computer, and the presented results are an average of 10 runs. Fig.5, Fig.6 and Fig.7 show the quality (the lower the execution time the higher the quality), and performance

(the lower the mapping time the higher the performance) of the experimented algorithms when mapping app-1, app-2 and app-3 on the testbeds, respectively.

In Fig.5, QMs are better than the other algorithms in terms of quality. QM-2, which coarsens the application graph, gives the better solutions than QM-1 while GA is the worst. Note that MiniMax-4 is not applicable since the number of tasks (i.e. $|V| = 100$) is less than the number of nodes in all testbeds. Also notice that selecting algorithms have effects on the quality of the solutions. Random selection (MiniMax-3) is the worst; however, it is not conclusive between MiniMax-1 and MiniMax-2. In terms of performance, GA takes much longer time than the other algorithms. MiniMax-3 is the fastest while QM-2 has only little overhead over MiniMax-3's. Notice that QM-1 is slower than QM-2 since QM-1 searches for the best node from all nodes in the environment.

In Fig.6, MiniMax-4 is applicable when mapping the application on grid-10 to grid-60 since the number of tasks (i.e. $|V| = 2500$) is more than the numbers of nodes in these testbeds. In terms of quality, QM-2 outperforms the other algorithms, and GA is the worst. In terms of performance, GA appears to be the worst. However, when mapping the application on grid-50 and grid-60, MiniMax-4 takes the longest time. This is due the large number of tasks (i.e. $|V| = 2500$) and the sharp increase in the number of nodes (from 1228 in grid-40 to 1679 and 1842 in grid-50 and grid-60, respectively). QM-2 is the fastest, and both QMs are faster than all MiniMax algorithms in all mapping cases.

In Fig.7, QM-2 still outperforms the other algorithms. In terms of quality, MiniMax-4 can produce the solutions with the quality close to QM-2's while GA

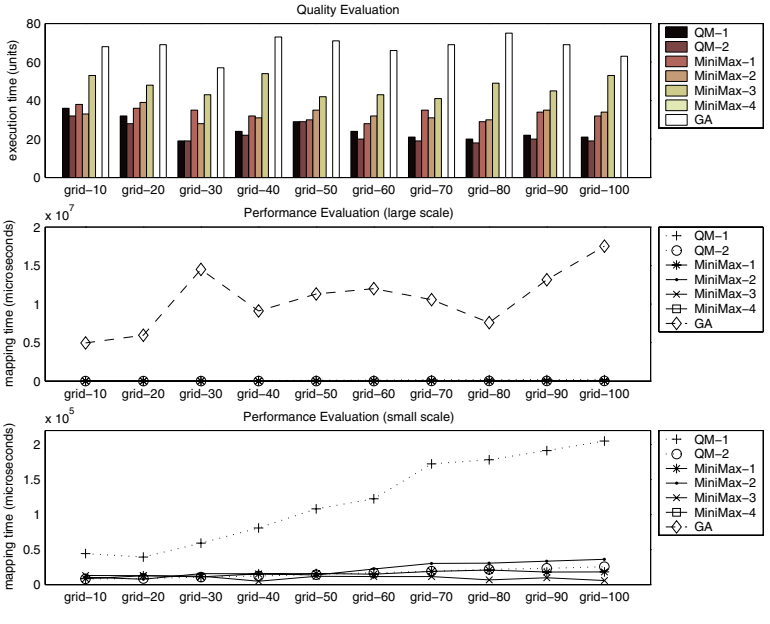


Fig. 5. The quality and performance of QM, MiniMax and GA when mapping app-1

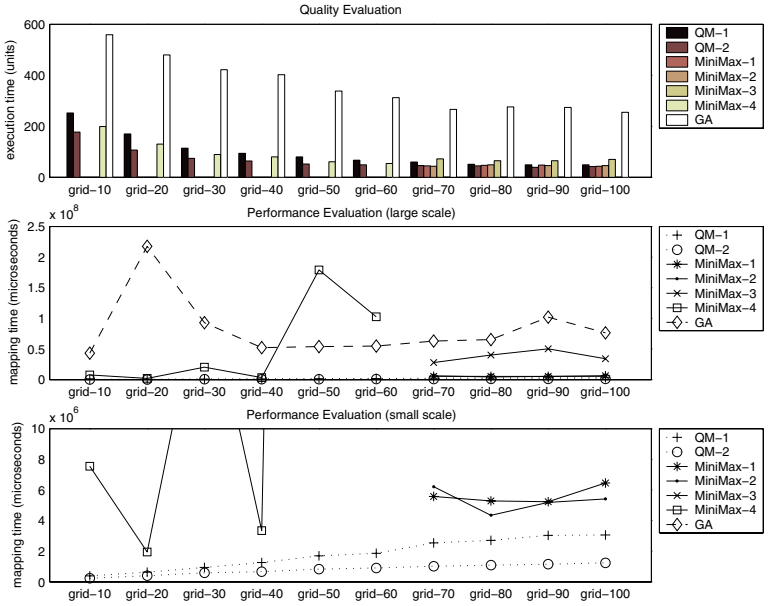


Fig. 6. The quality and performance of QM, MiniMax and GA when mapping app-2

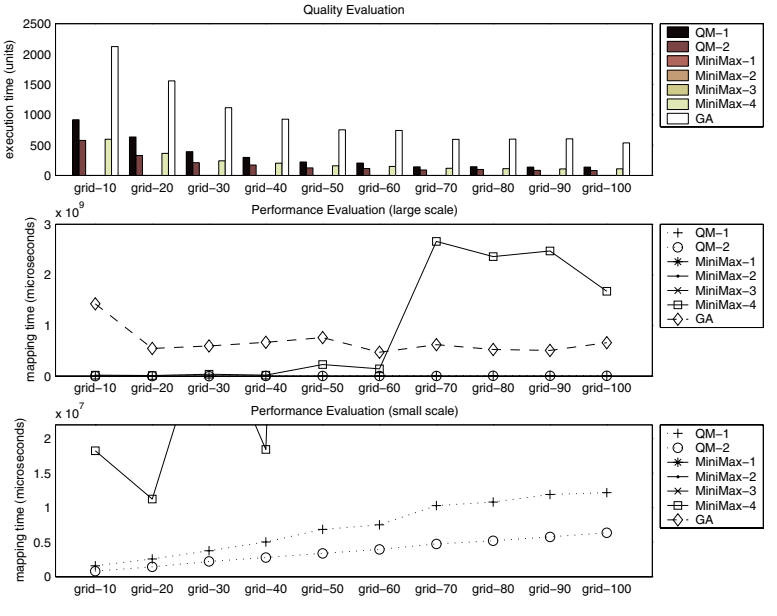


Fig. 7. The quality and performance of QM, MiniMax and GA when mapping app-3

is still the worst. In terms of performance, GA takes the longest time to map the application on grid-10 to grid-60 whereas MiniMax-4 is the slowest algorithm when mapping the application on grid-70 to grid-100. This is also due to the large number of tasks (i.e. $|V| = 10000$), and the sharp increase in the number of nodes (from 1843 in grid-60 to 2535 in grid-70). QM-2 is still the fastest.

When considering both the quality and the performance, QM-2 performs better than the other algorithms. For example, QM-2 can map app-3 (10000 tasks) on grid-100 (2992 nodes) in 6.35 seconds, and gives the lowest execution time while GA and MiniMax-4, respectively, take approximately 660 and 1700 seconds, but produce 6.60 and 1.34 times greater execution times than QM's. From the results of QM-1 and QM-2, it is conclusive that considering the best node according to proposition 1 improves the performance significantly while graph coarsening is a major key to improve the quality of the mapping solutions.

6 Conclusions

The existing algorithms to map parallel applications on computational grids, such as MiniMax, FastMap and genetic algorithms have shortcomings. In this paper, a new mapping algorithm is presented. The core idea is to map each task to a new node that gives a lower execution time to the task than its current node. The technique to coarsen the application graph is also employed. Experimental results show that QM performs better than the other algorithms, and graph coarsening is a major key to improve the quality of the mapping solutions.

Future work aims at deploying the algorithm in real environments. However, unrealistic assumptions hinder the deployment. These problems are currently being investigated [9].

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organisations. International Journal of Supercomputer Applications **15** (2001)
2. Bokhari, S.: On the mapping problem. IEEE Transaction on Computers **C-30** (1981) 207–214
3. Ullman, J.: Np-complete scheduling problems. Computer and System Sciences **10** (1975) 434–439
4. Kumar, S., Das, S., Biswas, R.: Graph partitioning for parallel applications in heterogeneous grid environments. In: International Parallel and Distributed Processing Symposium (IPDPS 2002). (2002) 66–72
5. Sanyal, S., Jain, A., Das, S., Biswas, R.: A hierarchical and distributed approach for mapping large applications onto heterogeneous grids using genetic algorithms. In: IEEE International Conference on Cluster Computing. (2003) 496–499
6. Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys **31** (1999)

7. Liu, C., Yang, L., Foster, I., Angulo, D.: Design and evaluation of a resource selection framework for grid applications. In: 11th IEEE International Symposium on High Performance and Distributed Computing (HPDC 2002), Edinburgh, Scotland (2002)
8. Braun, T., Siegel, H., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., Yao, B.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing **61** (2001) 810–837
9. Phinjaroenphan, P., Bevinakoppa, S., Zeephongsekul, P.: A method for estimating the execution time of a parallel task on a grid node. In: European Grid Conference. (2005)

A Bypass of Cohen's Impossibility Result

Jan A. Bergstra[1,2] and Alban Ponse[1]

[1] University of Amsterdam, Programming Research Group, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
[2] Utrecht University, Department of Philosophy, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

Abstract. Detecting illegal resource access in the setting of grid computing is similar to the problem of virus detection as put forward by Fred Cohen in 1984. We discuss Cohen's impossibility result on virus detection, and introduce "risk assessment of security hazards", a notion that is decidable for a large class of program behaviors.

Keywords: Malcode, Program algebra, Thread algebra, Virus, Worm.

1 Introduction

Grid computing poses many challenges which are known from computer science, though now at an integrated level. For semantic work about the grid it is hard to reach that level of integration as well. An example of semantic work on grid computing is in [8], in which the authors Nemeth and Sunderam point out that convincing definitions of what constitutes a grid are still hard to obtain. They set out to provide a definition, and in the preparation of the formal work a table is presented with a comparison of grids and conventional distributed systems. It is stated in their Table 1 that in a grid, access to a node may not imply access to all of its resources and that users working from another node may have little information on accessible resources.

We will study this aspect in particular under the assumption that a node takes the responsibility to prevent illegal access to resources by tasks it accepts to execute. We try to find a simplest possible model for this issue and use thread algebra in combination with program algebra. These techniques were introduced in [2] and [1], respectively. (The single thread version of thread algebra is named polarized process algebra, see [1]). This leads to the preliminary conclusion that a fair amount of phenomena in concurrent processing may be understood and formalized along those lines, in particular phenomena where non-determinism is immaterial. It seems to be the case that many characteristic aspects of grid computing can be analyzed using the paradigm of strategic interleaving as put forward in [2]. That means that instead of taking a combinatorial explosion of different runs into account, a limited portfolio of interleaving strategies may be used to characterize vital phenomena, including issues concerning thread mobility and access control. Deterministic strategies inherit from planning theory as discussed in [6].

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1097–1106, 2005.

It appears that detecting illegal resource access is formally very similar to the problem of virus detection as put forward by Cohen in 1984. There is a vast amount of literature on virus detection, and opinions seem to differ wildly. Many authors agree that malcode contains all others, and that both a virus and a worm can replicate. Furthermore, a worm is more autonomous than a virus. Some authors claim that a virus can only replicate as a consequence of actions of users, and that sound education and awareness can protect users from acting with such effect. So, a virus uses a user for its replication; that user may or may not be a victim of the virus' harmful action at the same time. Unclear is if each of these users must be a human one or if background processes in a machine can also be "used" as users.

This paper focuses on virus detection and discusses two fundamental questions. First, we consider Cohen's result about the impossibility of a uniform algorithm (or tool) for detecting (forecasting) viruses in programs [5]. This is done in the setting of the program algebra PGA [1]. Then, we define a different notion of testing — *security hazard risk assessment* — with which the occurrence of security hazards is decidable for a large class of program behaviors. However, if divergence (the absence of halting) is considered also as a security hazard, decidability is lost.

The paper is organized as follows: in Section 2 and 3 we introduce some basics of program algebra and the setting in which we will analyze code security risks. Then, in Section 4, we consider Cohen's impossibility result and some related issues. In Section 5 we introduce our notion of security hazard risk assessment. The paper is ended with some conclusions.

2 Basics of Program Algebra

Program algebra (PGA, [1]) provides a very simple notation for sequential programs and a setting in which programs can be systematically analyzed and mapped onto behaviors. Program behaviors are modeled in thread algebra. Finally, we consider some other program notations based on program algebra.

The Program Algebra PGA. In PGA we consider *basic* instructions $a, b, ...$ given by some collection B. Furthermore, for each $a \in B$ there is a *positive test* instruction $+a$ and a *negative test* instruction $-a$. The control instructions are *termination*, notation !, and (relative) jump instructions $\#k$ ($k \in \mathbb{N}$). Program expressions in PGA, or shortly PGA-programs, have the following syntax:

- each PGA-instruction is a PGA-program,
- if X and Y are PGA-programs, so is their *concatenation* $X; Y$,
- if X is a PGA-program, so is its repetition X^ω.

The behavior associated with the execution of PGA-programs is explained below. Instruction congruence of programs has a simple axiomatization, given in Table 1. The axioms PGA1-4 imply *Unfolding*, i.e. the law $X^\omega = X; X^\omega$, and PGA2-4 may be replaced by Unfolding and the proof rule $Y = X; Y \;\Rightarrow\; Y = X^\omega$.

Table 1. Axioms for PGA's instruction sequence congruence

$(X;Y);Z = X;(Y;Z)$	(PGA1)	$X^\omega;Y = X^\omega$	(PGA3)
$(X^n)^\omega = X^\omega$ for $n > 0$	(PGA2)	$(X;Y)^\omega = X;(Y;X)^\omega$	(PGA4)

Table 2. Equations for behavior extraction on PGA

$\lvert!\rvert = S$	$\lvert!;X\rvert = S$	$\lvert\#k\rvert = D$
$\lvert a\rvert = a \circ D$	$\lvert a;X\rvert = a \circ \lvert X\rvert$	$\lvert\#0;X\rvert = D$
$\lvert{+}a\rvert = a \circ D$	$\lvert{+}a;X\rvert = \lvert X\rvert \trianglelefteq a \trianglerighteq \lvert\#2;X\rvert$	$\lvert\#1;X\rvert = \lvert X\rvert$
$\lvert{-}a\rvert = a \circ D$	$\lvert{-}a;X\rvert = \lvert\#2;X\rvert \trianglelefteq a \trianglerighteq \lvert X\rvert$	$\lvert\#k{+}2;u\rvert = D$
		$\lvert\#k{+}2;u;X\rvert = \lvert\#k{+}1;X\rvert$

Thread Algebra. Execution of PGA-programs is modeled in thread algebra. Given B, now considered as a collection of *actions*, it is assumed that upon execution each action generates a Boolean reply (true or false). Now, behavior is specified in thread algebra by means of the following constants and operations:

Termination. The constant S represents (successful) termination.

Inaction. The constant D represents the situation in which no subsequent behavior is possible. (Sometimes the special thread D is called *deadlock* or *divergence*.)

Post Conditional Composition. For each action $a \in B$ and threads P and Q, the post conditional composition $P \trianglelefteq a \trianglerighteq Q$ describes the thread that first executes action a, and continues with P if `true` was generated, and Q otherwise.

Action Prefix. For $a \in B$ and thread P, the action prefix $a \circ P$ describes the thread that first executes a and then continues with P, irrespective of the Boolean reply. Action prefix is a special case of post conditional composition: $a \circ P = P \trianglelefteq a \trianglerighteq P$.

Behavior Extraction: From Program Algebra to Thread Algebra. The behavior extraction operator $\lvert X\rvert$ assigns a behavior to program X. Instruction sequence equivalent programs have of course the same behavior. Behavior extraction is defined by the thirteen equations in Table 2, where $a \in B$ and u is a PGA-instruction.

Some examples: $\lvert(\#0)^\omega\rvert = \lvert\#0;(\#0)^\omega\rvert = D$ and, further taking action prefix to bind stronger than post conditional composition,

$$\begin{aligned}\lvert{-}a;b;c\rvert &= \lvert\#2;b;c\rvert \trianglelefteq a \trianglerighteq \lvert b;c\rvert\\ &= \lvert\#1;c\rvert \trianglelefteq a \trianglerighteq b \circ \lvert c\rvert\\ &= \lvert c\rvert \trianglelefteq a \trianglerighteq b \circ c \circ D\\ &= c \circ D \trianglelefteq a \trianglerighteq b \circ c \circ D.\end{aligned}$$

In some cases, these equations can be applied (from left to right) without ever generating any behavior, e.g.,

$$|(\#1)^\omega| = |\#1;(\#1)^\omega| = |(\#1)^\omega| = \ldots$$
$$|(\#2;a)^\omega| = |\#2;a;(\#2;a)^\omega| = |\#1;(\#2;a)^\omega| = |(\#2;a)^\omega| = \ldots$$

In such cases, the extracted behavior is defined as the thread D.

It is also possible that behavioral extraction yields an infinite recursion, e.g.,

$$|a^\omega| = |a;a^\omega| = a \circ |a^\omega|,$$

and therefore, $|a^\omega| = a \circ |a^\omega| = a \circ a \circ |a^\omega| = a \circ a \circ a \circ |a^\omega| \cdots$. In such cases the behavior of X is infinite, and can be represented by a finite number of behavioral equations, e.g., $|(a;+b;\#3;-b;\#4)^\omega| = P$ and

$$P = a \circ (P \trianglelefteq b \trianglerighteq Q),$$
$$Q = P \trianglelefteq b \trianglerighteq Q.$$

The program notations PGLB and PGLC. The program notation PGLB is obtained from PGA by adding backwards jumps $\backslash\#k$ and leaving out the repetition operator. For example, the thread defined by PGLB-program $+a;\backslash\#1;+b$ behaves as $(+a;\#4;+b;\#0;\#0)^\omega$, i.e., as P in $P = P \trianglelefteq a \trianglerighteq b \circ D$.

This is defined with help of a projection function `pglb2pga` that translates PGLB-programs in a context-dependent fashion. For a PGLB-program X we write $|X|_{pglb} = |\texttt{pglb2pga}(X)|$ (see further [1]).

The language PGLC is the variant of PGLB in which termination is modeled implicitly: a program terminates after its last instruction has been executed and that instruction was no jump into the program, or it terminates after a jump outside the program. The termination instruction ! is not present in PGLC. For example,

$$\begin{aligned} |+a;\#2;\backslash\#2;+b|_{pglc} &= |+a;\#2;\backslash\#2;+b;!;!|_{pglb} \\ &= |(+a;\#2;\#6;+b;!;!;\#0;\#0)^\omega| \\ &= P \end{aligned}$$

for $P = b \circ S \trianglelefteq a \trianglerighteq P$ (see [1] for precise definitions of $|X|_{pglc}$ and $|Y|_{pglb}$.)

3 Detecting Access to a Forbidden Resource

In this section we introduce the setting in which we will analyze code security risks. We now consider a thread algebra with actions in "focus-method" notation, i.e., actions of the form $f.m$ where f is the *focus* and m the *method*. A *forbidden resource* is a resource that may not be accessed by threads of ordinary security clearance. A focus containing a forbidden resource is called a high risk focus. The state of affairs in which a thread plans to access a forbidden resource is called a *security hazard*.

Let P be some thread that uses communication with the following typical resources H_e, H_f and H_g:

$\boxed{\boxed{P}} \xrightarrow{e} \boxed{H_e}$ (external focus)

$f \downarrow \quad \searrow g \; \boxed{H_g}$ (low risk focus, no security hazard)

$\boxed{H_f}$ (high risk focus, security risk)

The reply of a basic instruction $e.m$ will be determined by the resource H_e. Likewise, instructions with focus f or g communicate with H_f and H_g, respectively.

Now, execution is *secure* if no $f.m$ is called until termination or first call of some $e.m$ (to the external focus).

A thread can have low risk actions (secure execution expected) and high risk actions (insecure execution expected). For example,

S — a low risk behavior (no security hazard),
$f.m \circ S$ — a high risk behavior (security hazard),
$f.m \circ S \trianglelefteq g.m \trianglerighteq g.m \circ S$ — risk depends on H_g (potential security hazard).

Suppose in some network, a site C receives the description p in some programming language PGLX of a thread $P = |p|_{pglx}$ to be run at C. Then

$$P \trianglelefteq \mathit{sctest.ok} \trianglerighteq S$$

formalizes a way for C to run P only if its security has been cleared: the test action *sctest.ok* (security clearance test) models this type of testing, yielding true if P is secure. In terms of the above modeling, such type of testing may be performed by a secure resource like H_g with focus *sctest* (thus H_{sctest}).

Alternatively, one can consider a test resource $H_{asctest}$ (alternative security clearance test) which produces true in

$$P \trianglelefteq \mathit{asctest.ok} \trianglerighteq S$$

if P has a security hazard. This resource may be implemented by always returning false. Consequently, the better option is to require that if in

$$P \trianglelefteq \mathit{asctest.ok} \trianglerighteq Q$$

the test *asctest.ok* yields false, the security of thread Q is guaranteed. In the next section we show that such a seemingly natural test action is self-contradictory; in Section 5 we propose a variant of *sctest.ok* that is not manifestly self-contradictory.

4 Security Hazard Forecasting

In this section we consider a security hazard forecasting tool and establish a formal correspondence between security hazard detection (a thread plans to access a forbidden resource) and the virus detection problem put forward by Fred Cohen in 1984.

Let SHFT be a Security Hazard Forecasting Tool with focus *shft*, thus a resource that forecasts a security hazard. As assumed earlier, a security hazard is in our simple setting a call (action) $f.m$ for some m. Furthermore, let *shft.test* be the test that uses SHFT in the following way: in

$$P \trianglelefteq \mathit{shft.test} \trianglerighteq Q,$$

the action *shft.test* returns true if P has a security hazard, and false if Q has no security hazard.

Theorem 1. *A Security Hazard Forecasting Tool cannot exist.*

Proof. Consider $S \trianglelefteq \mathit{shft.test} \trianglerighteq f.m \circ S$. If the test action *shft.test* returns false, then $f.m \circ S$ will be performed, which is a security hazard; if true is returned, then S is performed and no security hazard arises. □

The particular thread used in the proof above illustrates the impossibility of predicting that a thread (or a program) contains a virus, a general phenomenon that was described in Cohen's famous 1984-paper [5] and that will be further referred to as *Cohen's impossibility result*. For the sake of completeness, we recall Cohen's line of reasoning. In the pseudo-code below (taken from [5]), `D` is a decision procedure that determines whether a program is (contains) a virus, `~D` stands for its negation, and `next` labels the remainder of some (innocent) program:

```
program contradictory-virus:=
{1234567;

subroutine infect-executable:=
 {loop:file = get-random-executable-file;
 if first-line-of-file = 1234567 then goto loop;
 prepend virus to file;
 }

subroutine do-damage:=
 {whatever damage is to be done}

subroutine trigger-pulled:=
 {return true if some condition holds}
```

```
main-program:=

 {if ~D(contradictory-virus) then
    {infect-executable;
    if trigger-pulled then do-damage;
    }
 goto next;
 }
}
```

In PGLC, the program `contradictory-virus` can be represented by the following term `CV`:

$$\texttt{CV} = \#8; \texttt{Pre}; \#3; -\mathit{shft}.\texttt{test}(\texttt{CV}); \backslash\#8; \texttt{Next}$$

where `Pre` abbreviates the six instructions that model the security hazard:

$$\begin{aligned}\texttt{Pre} = {} & \texttt{file:=get-random-executable-file};\\ & \texttt{+first-line-of-file=1234567}; \backslash\#2; \texttt{prepend};\\ & \texttt{+trigger-pulled}; \texttt{do-damage}\end{aligned}$$

and `Next` models the remainder of the program. Applying behavior extraction on this program yields

$$\begin{aligned}|\texttt{CV}|_{pglc} &= |\texttt{Next}|_{pglc} \trianglelefteq \mathit{shft}.\texttt{test}(\texttt{CV}) \trianglerighteq |\texttt{Pre}; \#3; -\mathit{shft}.\texttt{test}(\texttt{CV}); \backslash\#8; \texttt{Next}|_{pglc}\\ &= |\texttt{Next}|_{pglc} \trianglelefteq \mathit{shft}.\texttt{test}(\texttt{CV}) \trianglerighteq |\texttt{Pre}; \texttt{Next}|_{pglc}\end{aligned}$$

So, $S \trianglelefteq \mathit{shft.test} \trianglerighteq f.m \circ S$ is indeed a faithful characterization of Cohen's impossibility result.

We note that even with the aid of universal computational power, the problem whether a thread has a security hazard (issues an $f.m$ call) is undecidable. This problem can be seen as a variant of the unsolvability of the Halting Problem, i.e., Turing's impossibility result.

Cohen's impossibility result needs the notion of a secure run (no security hazards), as well as a secure program or behavior (a thread that will have secure runs only). So, Cohen's impossibility result emerges if:

- secure runs exist,
- secure threads exist,
- there is a full match between these two,
- forecasting is possible.

Now there is a difficulty with forecasting: if *shft.test* returns false one hopes to proceed in such a way that the security hazard is avoided (why else do the test?). But that is not sound as was shown above. Thus we conclude: this type of security hazard forecasting is a problematic idea for the assessment of security hazards.

5 Security Hazard Risk Assessment

In this section we introduce a *security hazard risk assessment* tool, taking into account the above-mentioned considerations. This tool turns out to be a much more plausible modeling of testing the occurrence of security hazards. However, if we add divergence (the absence of halting) as a security risk, the tool can not exist.

The following security hazard risk assessment tool SHRAT with focus *shrat* may be conceived of as assessing a security hazard risk. In

$$P \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$$

the test action *shrat.ok* returns true if P is secure, and false if P is insecure (then P is avoided and Q is done instead). This is a more rational test than *shft.test* because it tests only a single thread (its left argument). Using an external focus e, the test action *shrat.ok* in

$$(P_1 \trianglelefteq e.m \trianglerighteq P_2) \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$$

yields true because $e.m$ is seen as an action that is beyond control of security hazard risk assessment.

For testing *shrat.ok* actions we can employ backtracking: at $P \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$,

1. Temporarily remove thread (or loaded program),
2. Place P instead
3. Execute[1] until $\begin{cases} S \text{ or } D \text{ or } e.m & \Rightarrow \text{backtrack if possible, otherwise } \mathsf{true}, \\ f.m & \Rightarrow \text{backtrack if possible, otherwise } \mathsf{false}, \\ P' \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q' & \Rightarrow \text{restart 1 with } P' \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q', \end{cases}$

The backtracking in this algorithm may require the testing of threads that are no direct subthreads of the original one, e.g., in

$$(P_1 \trianglelefteq \mathit{shrat.ok} \trianglerighteq P_2) \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$$

first the leftmost *shrat.ok* action is evaluated. If this yields false (so P_1 contains a security hazard), then $P_2 \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$ is evaluated. For finite threads this is a terminating procedure and not problematic.

Evaluation of *shrat.ok* actions can be extended to a larger class of threads. A *regular* thread P_1 over B is defined by a finite system of equations over $\overline{P} = P_1, ..., P_n$ (for some $n \geq 1$) of the following form:

$$\begin{aligned} P_1 &= F_1(\overline{P}) \\ &\vdots \\ P_n &= F_n(\overline{P}) \end{aligned}$$

with $F_i(\overline{P}) ::= S \mid D \mid P_{i,1} \trianglelefteq a_i \trianglerighteq P_{i,2}$ for $P_{i,j} \in \{P_1, ..., P_n\}$ and $a_i \in B$. Consider $P_1 \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$, thus $F_1(\overline{P}) \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$. Again we can decide the outcome

[1] Here "execute" means that upon a test action a, both branches should be inspected.

of the test action *shrat.ok* by doing a finite number of substitutions, linear in n. (Loops and divergence are not considered security hazards.) This leads to the following result:

Theorem 2. *For regular threads, the tool SHRAT is possible.*

We give a simple example: if

$$P_1 = P_2 \trianglelefteq a \trianglerighteq P_1$$
$$P_2 = P_1 \trianglelefteq f.m \trianglerighteq P_1 \quad (= f.m \circ P_1),$$

then *shrat.ok* in $(P_2 \trianglelefteq a \trianglerighteq P_1) \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$ yields true if it does in both $P_1 \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$ and $P_2 \trianglelefteq \mathit{shrat.ok} \trianglerighteq Q$. Obviously, it does not in the latter case, so this thread equals Q. A slightly more complex example (including the evaluation of the various *shrat.ok* tests):

$$P_1 = P_2 \trianglelefteq \mathit{shrat.ok} \trianglerighteq S \qquad (\mathsf{true})$$
$$P_2 = P_3 \trianglelefteq a \trianglerighteq P_4$$
$$P_3 = P_2 \trianglelefteq \mathit{shrat.ok} \trianglerighteq P_6 \qquad (\mathsf{true})$$
$$P_4 = P_7 \trianglelefteq \mathit{shrat.ok} \trianglerighteq P_8 \qquad (\mathsf{false})$$
$$P_5 = a \circ P_2$$
$$P_6 = f.m \circ P_2$$
$$P_7 = f.m \circ P_2$$
$$P_8 = a \circ S.$$

Omitting the *shrat.ok*-tests, thread P_1 has behavior P as defined in

$$P = a \circ P \trianglelefteq a \trianglerighteq a \circ S.$$

Thus, evaluation of the reply of *shrat.ok* is decidable for regular threads. We conclude that Cohen's impossibility result does not apply in this case; apparently, that result is about forecasting. Of course, the decidability of the evaluation of *shrat.ok* actions is lost if a Turing Tape is used as a resource.

Divergence Risk Assessment. If we consider divergence as a security hazard, say by focus *drat* and resource DRAT (a Divergence Risk Assessment Tool), we have a totally different situation: in the thread defined by

$$P = P \trianglelefteq \mathit{drat.ok} \trianglerighteq S$$

we then obviously want that the test action *drat.ok* returns the answer false. It is well-known that in general, DRAT cannot exist, as it is equivalent with solving the Halting Problem.

Now, involving divergence as a security hazard in *shrat.ok* actions, we also find that in

$$P = P \trianglelefteq \mathit{shrat.ok} \trianglerighteq f.m \circ S$$

the test should yield false (otherwise divergence). However, this yields a problem: in

$$P = P \trianglelefteq \mathit{shrat.ok} \trianglerighteq S$$

this goes wrong: the halting problem (Turing's impossibility result) wins, and hence the backtracking model is not suitable anymore.

6 Conclusions

In [3], we provide a formal treatment of the (potential) interaction of a thread P with resources H_e, H_f and H_g. Notation for that situation is

$$((P/_e H_e)/_f H_f)/_g H_g \text{ or equivalently, } P/_e H_e/_f H_f/_g H_g.$$

In the present paper we considered all communications of P with a resource H_h implicit and wrote P instead. In other words, an expression like $P \trianglelefteq h.m \trianglerighteq Q$ as occurring in this paper is considered to abbreviate

$$(P \trianglelefteq h.m \trianglerighteq Q)/_h H_h,$$

and this type of interaction is formalized in [3]. In [4] we provide a process algebraic semantics of threads $(P \trianglelefteq h.m \trianglerighteq Q)/_h H_h$.

In [7] it is stated that in order to constitute a grid, a network must implement the Globus GRAM protocol. This is a far more constrained definition than the concept based definitions that occur in [8]. We do not use that characterization because it is too complex for a brief theoretical paper.

How do our results relate to GRAM? Secure resource allocation on the grid requires an underlying theoretical basis. What we put forward is that under a very simple definition of a resource allocation risk, Cohen's impossibility result need not constrain the options for automatic detection as much as one might think. On the contrary, if security risk avoidance is adapted as a strategy, Cohen's impossibility result disappears.

References

1. J.A. Bergstra and M.E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002.
2. J.A. Bergstra and C.A. Middelburg. Thread algebra for strategic interleaving. Computing Science Report 04-35, Eindhoven University of Technology, Department of Mathematics and Computing Science, November 2004.
3. J.A. Bergstra and A. Ponse. Combining programs and state machines. *Journal of Logic and Algebraic Programming*, 51(2):175–192, 2002.
4. J.A. Bergstra and A. Ponse. Execution architectures for program algebra. Logic Group Preprint Series 230, Dept. of Philosophy, Utrecht University, 2004.
5. F. Cohen. Computer viruses - theory and experiments, 1984. `http://vx.netlux.org/lib/afc01.html`. Version including some corrections and references: *Computers & Security* 6(1): 22-35, 1987.
6. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1:25-39, 2003.
7. S. Hwang and C. Kesselman. A flexible framework for fault tolerance in the grid. *Journal of Grid Computing*, 1(3):251-272, 2003.
8. Zs. Nemeth and V. Sunderam. Characterizing grids: attributes, definitions, and formalisms. *Journal of Grid Computing*, 1:9-23, 2003.

Mapping Workflows onto Grid Resources Within an SLA Context

Dang Minh Quan and Odej Kao

Department of Computer Science, University of Paderborn, Germany

Abstract. The mapping of workflows on Grid resources differs from mapping a single job, as dependencies between the subjobs has to be considered and resolved. This is a major task, if underlying Service Level Agreements (SLAs) have to be satisfied, which define the deadline for the entire workflow but also allow flexibility while assigning the subjobs to resources. Therefore, new requirements regarding selection of optimal resource destination and satisfying multiple time constraints are important issues, which are not completely addressed by existing methods.

This paper presents a method which performs an efficient and precise assignment of workflow subjobs to Grid resources with respect to SLAs and subjob dependencies. The quality of the created results is evaluated with a number of experiments and compared to the results of existing planning tools.

1 Introduction

The mapping of jobs to suitable resources is one of the core tasks in Grid Computing. A substantial research in this area led to a number of methods, which allow the specification of job requirements, the description of available resources and the matching in order to find an appropriate execution platform [1, 2, 3]. However, the majority of the research is related to the mapping of singular jobs, which do not exhibit dependencies to other jobs regarding input/output data. The mapping of workflows, where a single job is divided into several subjobs, is the next research step. Demands for Grid-based workflows result from many scientific and business application, where data or suitable resources are distributed over multiple administrative domains. Thus, supporting workflows in Grid environments increases the applicability of Grid Computing for a large number of processes.

An intuitive approach of treating each subjob as an independent job and its mapping with the traditional methods reaches the limits in case of SLA-based processing. An SLA defines the start and end time of the workflow execution, which has to be considered within the execution chain. Thus, the subjobs has to be mapped in a way, that the final workflow deadline will be reached, regardless of the actual execution resource for the subjobs and the time needed for the transmission of output/input data between the subjobs. Thereby, two main problems arise. Firstly, for each subjob involved in the workflow the start and

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1107–1116, 2005.

end time has to be computed and verified with respect to the global deadline. According to results the corresponding time slots on the selected resources have to be reserved. For this purpose we assume that the user provides the maximal runtime of each subjob and that the underlying resource management system (RMS) supports advance reservations such as CCS [5]. Queuing-based RMS are not suitable, as no information about the starting time is provided. Secondly, the selection of the Grid resources should be based on a minimal cost for the resource usage. For this purpose well-known models such as the flexible job shop scheduling [4, 7] are applied, which minimize the runtime of the workflow. In this paper, time is computed in slots, where each slot corresponds to a-priori defined time period. Reserved resources include number of CPUs, required memory size, availability of experts etc. An extension to other devices, software licenses, and other related resources is straightforward.

This paper is organized as follows. Section 2 presents a formal problem statement. Section 3 and 4 describe the related work and the proposed algorithm respectively. Empirical measurements and comparisons with existing planning methods are subject of Section 5.

2 Formal Problem Statement

The formal specification of the described problem includes following elements:

- Let K be the set of Grid RMSs. This set includes a finite number of RMSs which provide static information about controlled resources and the current reservations/assignments.
- Let S be the set of subjobs in a given workflow including all subjobs with the current resource and deadline requirements.
- Let E be the set of connections (edges) in the workflow, which express the dependency between the subjobs and the necessity for data transfers between the jobs.
- Let T_i be the set of time slots for the subjob $S_i, S_i \in S$.
- Let K_i be the set of resource candidates of subjob S_i . This set includes all resources which can run subjob S_i, $K_i \in K$.

Based on the given input, a feasible and possibly optimal solution is sought, which allows the most efficient mapping of the workflow in a Grid environment with respect to the given global deadline. The required solution is a set defined as

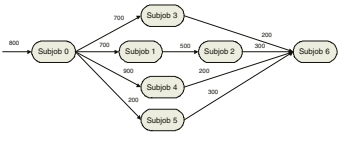


Fig. 1. A sample workflow

Table 1. RMSs resource reservation

ID	ID_hpc	CPUs	mem	exp	start	end
31	2	128	256000	8	0	1000000
23	0	128	256000	9	0	1000000
30	1	128	256000	6	0	1000000

Table 2. Resource requirements for subjobs

Sj_ID	CPU	mem	exp	runtime	earliest	latest
0	51	59700	1	21	5	35
1	62	130030	3	45	27	57
2	78	142887	4	13	57	87
3	128	112933	4	34	28	66
4	125	171354	2	21	28	65
5	104	97560	3	42	27	62
6	45	117883	1	55	71	101

Table 3. Connections

From	To	Transfer_time	Data
0	1	1	636
0	3	2	638
0	4	2	892
0	5	1	192
1	2	2	401
2	6	1	300
3	6	1	200
4	6	2	200
5	6	1	271

$$R = \{(S_i, k_{ij}, t_{il}) | S_i \in S, k_{ij} \in K_i, t_{il} \in T_i\} \tag{1}$$

A feasible solution must satisfy following conditions:

- For all $K_i \neq 0$ at least one RMS exists which can satisfy all resource requirements for each subjob.
- The earliest start time slot of the subjob $S_i \leq t_{il} \leq$ the latest start time slot of S_i. Each subjob must have its start time slot in the valid period.
- The dependencies of the subjobs are resolved and the execution order remains unchanged.
- Each RMS provides a profile of currently available resources and can run many subjobs of a single flow both sequentially and parallel. Those subjobs which run on the same RMS form a profile of resource requirement. With each RMS k_{ij} running subjobs of the Grid workflow, with each time slot in the profile of available resources and profile of resource requirements, the number of available resources must be larger than the resource requirement.

In the next phase the feasible solution with the lowest cost is sought. The cost of a Grid workflow in this specific example is defined as a sum of four factors: compute time, memory usage, cost of using experts knowledge and finally time required for transferring data between the involved resources. If two sequent subjobs run on the same RMS, the cost of transferring data from the previous subjob to the later subjob neglected.

An example of a specific instance of the described problem is presented in Figure 1 and Tables 1-3. Figure 1 visualizes the sequence of subjobs, the corresponding parameter are summarized in Table 3, where the data to be transferred between subjobs as well as the estimated duration of each task is presented. Table 2 describes the resource requirements and the range of valid start time slots of each subjob. Information about the available resources of the three involved RMS, which will execute the flow, is presented in Table 1.

Considering the RMS's ability to run several subjobs in parallel and the evaluation of resource profiles increases the complexity of the flexible job shop

scheduling problem. It can be shown easily that the optimal mapping of Grid-based workflows as described above is a NP hard problem.

3 Related Work

Cao et al. presented an algorithm that maps each subjob separately on individual Grid resources [2]. The algorithm processes one subjob at a time, schedules it to a suitable RMS with start time slot not conflicting the dependency of the flow. The selection of the destination resources is optimised with respect to a minimal completion time. When applying this strategy to the specified problem, each subjob will be assigned separately to the cheapest feasible RMS. This strategy allows fast computation of a feasible schedule, but it does not consider the entire workflow and the dependencies between the subjobs.

The mapping of Grid workflows onto Grid resources based on existing planning technology is presented in [1]. This work focuses on coding the problem to be compatible with the input format of specific planning systems and thus transferring the mapping problem to a planning problem. Although this is a flexible way to gain different destinations, significant disadvantages regarding the time-intensive computation, long response times and the missing consideration of Grid-specific constraints appeared. The latter is the main cause that the suggested solutions often do not express the expected quality.

The described problem can also be seen as a special case of the well known job shop scheduling problem [4, 7]. For solving this problem, two main methods – complete and incomplete method – exist. A complete method explores systematically the entire search space, while the incomplete (non-exact) method examines as rapidly as possible a large number of points according to selective or random strategy. Local search is one of the most prominent examples for this approach, which is realized by a number of methods such as Tabu Search [11, 12], Simulated Annealing [8], GA [6] etc. However, with the appearance of resource profiles, the evaluation at each step in local search is very hard. If the problem is big with highly flexible variable, the classical searching algorithm need very long time to find a good solution.

4 Planning Algorithm for Workflows

The proposed algorithm for mapping workflows on Grid resources uses Tabu search to find the best possible assignment of subjobs to resources. In order to shorten the computation time caused by the high number of resource profiles to be analysed and by the flexibility while determining start and end times for the subjobs, several techniques for reducing the search space are introduced.

The core algorithm requires a specification of the workflow and subjob requirements as well as the description of the available resources as input. This information is necessary in order to resolve the dependencies in the workflow and is stored in a file. The properties of the involved RMS are contained in a database.

This input information is processed according to the following algorithm steps:

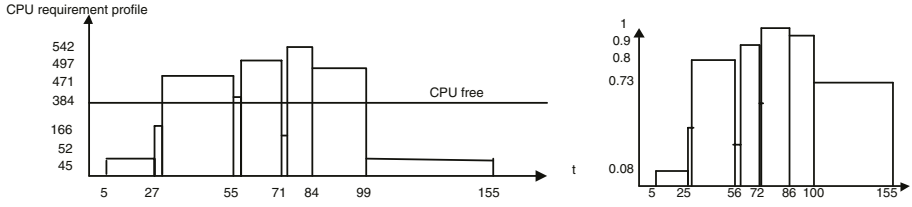
1. Based on the description in the database, all RMS are selected, which fulfil the requirements of at least one subjob.
2. Computation of the earliest start time and latest start time for each subjob by analysing the input workflow with traditional graph techniques.
3. Removing requirement bottlenecks. This task aims to detect bottlenecks with a large number of resource requirements, which can reduce the possible start/end times of a subjob. Based on this information subjobs are moved to other sites with more available resources in order to gain a longer time span for the positioning of the subjobs in the workflow.
4. Definition of the solution space by grouping all RMS candidates, which have enough available resources to start the subjobs within the determined slack time and to run the subjob until it is completed. This task is performed by establishing a connection with the site and retrieving the current and planned schedule.
5. The gained search space is evaluated with respect of the contained number of feasible solutions (large or small number of possible solutions). Subsequently, an initial solution is created.
6. Starting with the initial solution, a Tabu search is applied in order to increase the quality of the solution and to find the best possible and feasible assignment.

In following the individual algorithm steps are described in detail.

4.1 Resolving Requirement Bottlenecks

The generated requirement and resource profiles show those time slots, where a large number of resources is required, but not available. A sample of such situation is shown in Figure 2(a) on an example of required and available CPU instances. At each possible time slot, the total number of available CPUs over all involved RMS and the total number of required CPU as sum of the requirements of all subjobs possibly running in this time slot are computed. The contribution of each subjob in the profile is computed from the earliest start time to latest possible deadline. Figure 2(a) shows that at period 28-55 or 78-84 the number of required CPUs is larger than in other profile periods. This leads to a significantly reduced number of feasible start times and thus reduces the probability for finding an optimal solution or even a good solution. Therefore, the peak requirements have to be reduced by moving selected jobs to other time slots. Furthermore, by reducing the number of parallel running jobs on the same site, the probability for cost-effective execution of two subsequent jobs on the same site with a low communication effort increases.

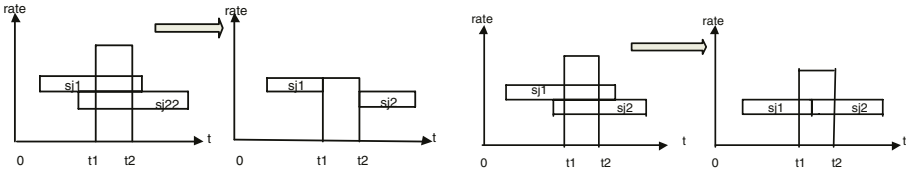
For resolving the requirement bottleneck the profiles of the required resources and the available resources are compared as shown in Figure 2(b). At each time slot, we define J as the set of m subjobs running at that time slot and R as the



(a) Relation between available and required CPUs

(b) Rate profile of the sample

Fig. 2. Profiles of the sample



(a) Moving subjobs

(b) Adjusting subjobs

Fig. 3. Removing possible requirement bottlenecks

set of n possible resource candidates for J. Subsequently, following measures can be computed.

$$TotalCPU_{require} := \sum_{i=1,m} J_i.CPU_{req} \quad \text{with} \quad J_i \in J \tag{2}$$

$$TotalMEM_{require} := \sum_{i=1,m} J_i.mem_{req} \quad \text{with} \quad J_i \in J \tag{3}$$

$$TotalEXP_{require} := \sum_{i=1,m} J_i.EXP_{req} \quad \text{with} \quad J_i \in J \tag{4}$$

$$TotalCPU_{avail} := \sum_{j=1,n} R_j.CPU_{avail}\frac{m_j}{m} \quad \text{with} \quad J_i \in J,\ m_j \leq m \tag{5}$$

$$Totalmem_{avail} := \sum_{j=1,n} R_j.mem_{avail}\frac{m_j}{m} \quad \text{with} \quad J_i \in J,\ m_j \leq m \tag{6}$$

$$TotalEXP_{avail} := \sum_{j=1,n} R_j.exp_{avail}\frac{m_j}{m} \quad \text{with} \quad J_i \in J,\ m_j \leq m \tag{7}$$

The parameter $J_i.CPU_{req}$, $J_i.mem_{req}$, and $J_i.EXP_{req}$ represent the number of required CPUs, the size of needed memory (in MB), the required experts for supervision of the of subjob J_i respectively. Finally, m_j is the number of subjobs which R_j can run simultaneously.

$$rate := \frac{\frac{TotalCPU_{require}}{TotalCPU_{avail}} + \frac{Totalmem_{require}}{Totalmem_{avail}} + \frac{TotalEXP_{require}}{TotalEXP_{avail}}}{3} \tag{8}$$

The removal of the requirement peak is performed by adjusting the start time slot or the end time slot of the subjobs and thus by moving out of the

bottleneck area. One possible solution is shown in Figure 3(a), where either the latest finished time of subjob1 is set to t_1 or the earliest start time of subjob2 is set to t_2. The second way is to adjust both subjobs simultaneously as depicted in Figure 3(b). A necessary prerequisite here is that after adjusting, the latest completion time - earliest start time is larger than the total runtime.

4.2 Initial Solution

The algorithm for determining the initial solution is based on a fail-first heuristic and the forward checking algorithm [10]. According to the Greedy strategy for each subjob under investigation, an RMS with the minimal cost is selected and assigned as described in the following four steps:

1. Determine the subjob in the workflow with the smallest number of RMS candidates, which can execute the subjob according to the provided specification.
2. If the set with RMS candidates for this job is empty, than assign one randomly selected RMS and mark the assignment as conflict. Otherwise, assign the RMS with the minimal cost to that subjob.
3. Repeat the process with the next subjob, until all subjobs are assigned.
4. Resolve the marked conflicts.

In case of conflicts, the Tabu search is modified in order to find a conflict-free and thus a feasible solution. The application of the Tabu search for finding at least one feasible solution is performed analogously with the one described in Section 4.3. Solely the cost function has to be replaced by function based on the number of remaining subjob conflicts in the workflow. If this is possible, the first found feasible solution is declared as initial solution and the mapping process proceeds with the Tabu search for the best solution. Otherwise the owner of the workflow is notified, that the workflow can not be executed according to the given conditions and rejected until new specification is submitted.

4.3 Tabu Search

The Tabu search is well-known method for finding feasible and best possible solutions for NP hard problems. In order to adapt Tabu search to the problem of workflow mapping, following parameters are set. The configuration $R_t \in R$ is any feasible assignment of the set $R = \{(S_i, k_{ij}, t_{il}) | S_i \in S, k_{ij} \in K_i, t_{il} \in T_i\}$. However, at this step for each S_i only one t_{il} is available leading to k_{ij} as a single variable factor. In the next step the neighbourhood to be evaluated is determined. Let R_t be a feasible configuration in R, then the neighbourhood $N(R_t)$ consists of all R'_t, where R'_t is a feasible configuration and R_t differs from R'_t at a single variable. The considered cost function is computed as stated in Section 2.

Subsequently, the created configuration is evaluated. In every iteration the most suitable candidate in the neighbourhood $N(R_t)$ is selected and the feasibility of the new configuration is checked. This is a compute-intensive step, as the

resource and job profiles (see Figure 2) have to be compared before determining the total processing cost. The Tabu tenure goes with each instance of resource in the solution space, thus after each move, the solution space is scanned and the Tabu list is updated. In order to consider very promising configurations, a simple aspiration criterion is applied. The Tabu status of a move is removed if the move leads to the solution with the lower cost than the best configuration found so far. The pseudo-code of the Tabu search algorithm is found in following:

```
begin {
  Get initial solution from previous step
  while(num_move < max){
     Select the best configuration from the current neighbourhood
     Update the Tabu list
     if (cost( ) > cost( ) )
          <---
     num_move+=1    }}
```

5 Performance Evaluation

Planning systems emerged as an effective and power tool for mapping Grid workflows to Grid resources[1]. Therefore, a sample planning-based method will be used as a basis for the comparison of the solution quality. A suitable data models were produced and given as input to a planning system and to the proposed algorithm. The planning method was selected from the list of systems, which participated in the international AI planning contests. Preliminary evaluations showed that solely Metric-FF [9] – well-known planner with very high performance – can handle fully the required Planning Data Description Language (PDDL 2.1) [13] and can solve the workflow mapping problem.

The hardware used for the experiments is rather standard and simple (Pentium 4 2,8Ghz, 1GB RAM, Linux Redhat 9.0). All necessary information about the resource requirements and resource specifications/reservations used in the experiments are on the web site wwwcs.upb.de/cs/ag-kao/en/persons/dang_minh/experiment1.html.

The goal of the experiment is to measure the feasibility of the solution, its cost and the time needed for the computation. For this purpose three different scenarios with increasing complexity level were analysed. The performance measurements started with a low-level experiment and a workflow with 7 subjobs and 3 RMSs as shown in Section 2. The total number of time slots is 138. The problem is coded in PDDL 2.1 and run using a simplified algorithm, where the feasibility check of resources is performed only at the start slot and the end slot of each subjob. The result of this experiment is presented in Table 4.

A second experiment is based on a Grid workflow presented in Figure 4, which includes 10 subjobs, 12 RMSs and 40 time slots. The results are presented in Table 5, where faster computation and slightly lower cost of the solution by the proposed algorithm were observed. Moreover, a significant difference in the

Table 4. Results of the simple level experiment

	Metric-FF		New method	
Subjob	RMS	TS start	RMS	TS start
0	RMS0	35	RMS2	5
1	RMS1	57	RMS2	27
2	RMS1	87	RMS2	57
3	RMS0	66	RMS1	28
4	RMS2	65	RMS1	65
5	RMS0	28	RMS0	62
6	RMS0	101	RMS2	101
Runtime	6.33 sec		< 1 sec	
Cost	171343.52		152870.61	

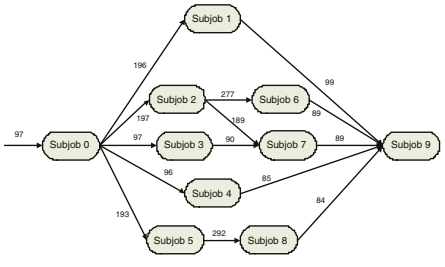


Fig. 4. Intermediate level flow

Table 5. Intermediate level flow

	Metric-FF		New Method	
Subjob	RMS	Start	RMS	Start
0	RMS5	14	RMS1	14
1	RMS0	36	RMS1	34
2	RMS0	33	RMS1	29
3	RMS5	34	RMS1	32
4	RMS5	21	RMS1	20
5	RMS0	21	RMS1	21
6	RMS3	40	RMS2	35
7	RMS0	37	RMS1	35
8	RMS0	32	RMS1	32
9	RMS0	43	RMS1	43
Runtime	67.13 sec		< 1 sec	
Cost	38802.96		38615	

required memory space was noticed, as Metric FF used about 500MB for the computation.

Finally, the last model contains 20 randomly selected jobs together with randomly selected requirements for CPU, memory, etc. Unfortunately, it was not possible to find a feasible solution with Metric-FF, as the existing memory was not sufficient while the proposed algorithm created a solution in a less than a second.

6 Conclusion

This paper presented a method which performs an efficient and precise assignment of workflow subjobs to Grid resources with respect to SLAs defined dead-

lines and subjob dependencies. The performance evaluation showed that the proposed algorithm creates solution of equal or better quality than well-known planning system Metric-FF and needs significantly shorter computation time and less main memory. The latter is a decisive factor for the applicability of the method in real environments, because large scale workflows can be planned and assigned efficiently.

Future work sets a strong focus on the network transfer rates, as the transfer time has a major impact on the possible starting and ending time slots for every subjob. If it is possible to predict the network performance, the planning process and the required reservations can be executed more precisely.

References

1. E. Deelman, J. Blythe, Y.Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda: Mapping Abstract Complex Workflows onto Grid Environments, Journal of Grid Computing, Vol 1, no. 1, pp. 25-39, 2003.
2. J. Cao, S. A. Jarvis, S. Saini, G. R. Nudd: GridFlow: Workflow Management for Grid Computing, Proc. 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Tokyo, Japan, pp. 198-205, 2003.
3. R. Lovas, G. Dzsa, P. Kacsuk, N. Podhorszki, D. Drtos: Workflow Support for Complex Grid Applications: Integrated and Portal Solutions, Proc. 2nd European Across Grids Conference, Nicosia, Cyprus, 2004.
4. J. W. Barnes, J. B. Chambers : Flexible Job Shop Scheduling by Tabu Search,Technical Report Series,Graduate Program in Operations Research and Industrial Engineering. The University of Texas at Austin, ORP96-09, 1996.
5. M. Hovestadt, O. Kao, A. Keller, A. Streit: Scheduling in HPC Resource Management Systems: Queuing vs. Planning, Proc. 9th Workshop on JSSPP at GGF8, LNCS 2862, pp. 1-20, 2003.
6. I. Kacem, S. Hammadi, P. Borne: Approach by Localization and Multi-objective Evolutionary Optimization for Flexible Job-Shop Scheduling Problems, IEEE Transactions on Systems, Man, and Cybernetics. Part C, Vol 32. N1, pp. 1-13, 2002.
7. S. Dauzere-Peres, J. Roux, J.B. Lasserre: Multi-resource shop scheduling with resource flexibility, European Journal of Operational Research, Vol 107, pp. 289-305, 1998.
8. N. Sadeh, Y. Nakakuki: Focused Simulated Annealing Search: An Application to Job Shop Scheduling, Annals of Operations Research, Vol 60, pp. 77-103, 1996.
9. J. Hoffmann: Extending FF to Numerical State Variables, Proc. 15th European Conference on Artificial Intelligence, Lyon, France, 2002.
10. V. Kumar: Algorithms for constraint-satisfaction problems: a survey, AI Magazine, Vol 13 n.1, pp.32-44, 1992.
11. F. Glover: Tabu search Part I, ORSA Journal on Computing, pp. 190-206, 1989.
12. F. Glover: Tabu search Part II, ORSAJournal on Computing, pp. 4-32, 1990.
13. M. Fox, D. Long: PDDL2.1: An extension of PDDL for expressing temporal planning domains, Journal of AI Research, Vol 20, pp. 61-124, 2003.

iShare - Open Internet Sharing Built on Peer-to-Peer and Web

Xiaojuan Ren and Rudolf Eigenmann

Purdue University, School of ECE,
West Lafayette, IN, 47907, USA
{xren, eigenman}@purdue.edu

Abstract. This paper presents design concepts and implementation overview of an Internet-sharing system, iShare. iShare supports end users as well as providers of Internet resources in disseminating, accessing and using these resources, in a way that allows open participation. A fully decentralized organization allows providers to simply post their resources on any web page, imposing no restrictions on resources attributes, administrative rules, and access protocols. Underneath its user surface it employs peer-to-peer information dissemination, advanced resource matching, open migration, and automatic service portal mechanisms. In addition to the qualitative comparison with related work, we evaluate the system in terms its efficiency of resource discovery and job execution.

1 Introduction

Internet-sharing systems harness the rapidly growing, worldwide resources of computer, network, and information systems. Advanced sharing technology bears tremendous potential in creating synergy among the users – both providers and end users – of machine platforms, networks, computer applications, software services, and information. We envision that future end users will be able to quickly learn about the availability of world-wide resources and to employ them as if they were located nearby, without download, installation, or maintenance effort. Providers of resources will be able to easily offer new software, hardware, or data to the community. In doing so, they will be able to define their own rules and create business or open source models, akin to today's economic principles.

In this paper we present design concepts and implementation overview of an Internet-sharing system, iShare[1], which serves as a research platform in pursuit of this vision. iShare facilitates sharing of three types of resources: programs (software tools, computational applications), machines (computers, devices), and data (documents, data bases). While much technology exists today for the sharing of data, the focus of iShare's initial thrust is on technology for sharing executable programs (a.k.a. *services* – we will use the two terms interchangeably) and their underlying compute platforms.

[1] It is based upon work supported in part by the U. S. National Science Foundation under Grants No. 9974976-EIA, 0103582-EIA, and 0429535-CCF.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1117–1127, 2005.

iShare extends the concepts of the PUNCH [7, 6] network computing system, which became operational in 1995 at Purdue University and has since served a large user community (over 3000 users in 35 countries) in computational nanotechnology, computer architecture and parallel programming. Both iShare and PUNCH have a strong end-user orientation. They 1) provide the means to disseminate, access and use networked resources and 2) they populate the infrastructure with services for the community. This orientation distinguishes the systems from related work in areas such as Grid computing and Web services. While these areas are pursuing goals similar to the original PUNCH project, their current focus is on developing the underlying technology, programming support, and standards for exploiting computational resources. iShare was motivated by an observed limitation of PUNCH, which it shares with many related efforts: populating the infrastructure with additional resources is limited by eligibility requirements and administrative procedures. For example, machines may only be added if they adopt certain administrative procedures; and programs may only be added after they become "grid-enabled". Furthermore, the new resources may need to be registered in or approved by a central organization before they become visible to the community. Removing these barriers would enable Internet resources to be shared in a truly open and scalable manner.

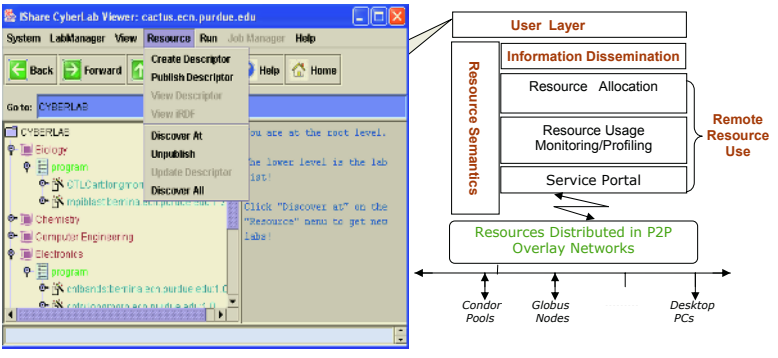
iShare addresses these issues by taking an *open publication* approach, which imposes no restrictions on participating resources. A web-posting mechanism publishes new resources and a fully distributed peer-to-peer mechanism supports resource discovery. Powerful resource matching mechanisms serve to map program resources onto fitting platforms. While iShare builds on existing standards and middleware technology, *protocol plug-ins* allow new protocols and standards to be added, satisfying the needs of resources posted in the future.

The remainder of this paper is organized as follows. Section 2 presents the design concepts and implementation of iShare. Section 3 provides evaluation results. We discuss related work in Section 4, followed by conclusions in Section 5.

2 The iShare Architecture

An Internet-sharing system can be characterized in terms of 1) its user model, 2) the semantics and type of the resources it can support, 3) the information disseminated about resources and activities at remote sites, and 4) the mechanisms that support the use of these resources at remote sites. This section elaborates on iShare's design concepts and implementations in these four areas.

The system architecture is shown in Figure 1(b). For space reasons, several important iShare components were left out of this paper – most notably iShare's authentication and trust mechanism (building on state-of-the-art techniques) as well as facilities for exchanging experience and bridging across diverse user communities. An extended version of this paper describes these components [5].



(a) The End User View in iShare (b) The iShare Architecture

Fig. 1. The End User View and the Structure of iShare

2.1 User Model

Supporting a *Resource Provider* User Class. Internet-sharing systems provide their user communities with facilities to discover and make use of resources created and maintained at remote sites. In addition to its *end users*, iShare considers *providers* of resources as an important user class. Resource developers take advantage of the ease with which resources can be made available to end users.

The basic iShare functionality for resource providers supports 1) creating resource descriptors, 2) publishing and removing resources, and 3) optionally starting and configuring advanced iShare services. Publication of a new resource is fully autonomous and does not require user interaction with any iShare "administrator". To end users, the published resources appear organized into discipline-specific *Cyberlabs*. Providers define (as a resource attribute) the Cyberlabs in which they wish to "place" the new resource. The core functionality for end users is to run the discovered, remote programs, using local files and displays as input/output.

Providing iShare Functionality Within Common Work Environments. Web portals provide convenient user interfaces to many Internet-sharing systems. However, advanced users often prefer to work within their common environment, such as a Unix Shell or Windows desktop. The iShare user functionality is accessible via an API, which allows different user interfaces to be built. Our initial design includes a Windows-based interface, as shown in Figure 1(a) and we are exploring iShare Unix Shell functionality.

2.2 Resource Semantics and Types

Resource Description with RDF. An important issue in Internet-sharing systems is the clear description of resources. Resource providers must define the semantics about what is shared, who is allowed to share, and how sharing occurs.

iShare adopts RDF as the resource description language for consistent encoding of resource attributes. RDF aims to specify semantics for data in a standardized interoperable manner [1] and provides a rich data model for describing objects.

iShare supports three types of resources at an equal level: software services (programs), service platforms (machines), and data. An important program attribute is that of a *pinned* or *roaming* service. Pinned services are applications running only on a specific platform, whereas roaming applications can execute on any matching platform. Machines are published as available hosts for roaming services. They might be individual machines or locally managed sub-systems (e.g., a network of workstations managed by Condor [9]).

Autonomy in Defining Resource Attributes. An important design concept is that resources can define their own rules of usage. Thus, a machine may offer services under its own administrative procedures and access protocols. iShare will find matching programs and platforms. Protocols that are not available initially may be plugged in, making the system incrementally more powerful.

To support the autonomy in defining local access rules, the standardized metadata definition must be extensible enough to support plug-in protocols. In our initial design, we formalized the metadata to include the common attributes of access protocols in widely used local management systems such as Condor [9] and PBS [10]. The metadata includes recompiling/relinking operations, submission syntax and user-commands. For example, accessing and utilizing a Condor pool involves relinking programs to utilize advanced features like checkpointing, creating a "submission files" with predefined syntax, and starting job running via specific commands: condor_submit. All this information must be described as resource access protocols. The information is then used for creating service portals, which hide the details from end users.

2.3 Information Dissemination

Decentralized Organization Built on P2P and Web. The information to support network-accessible computing is incremental and distributed in the sense that it is created from items at different locations and items available at different points in time. The incremental and distributed nature of the information makes it infeasible to collect and maintain it at a centralized location in a scalable manner. Thus, a decentralized scheme is essential for reliable and efficient information management. A key idea of iShare's decentralized structure is that a provider of resources can post their availability on any web page. Resource metadata is derived from these postings and inserted into a P2P network. While the World Wide Web affords unprecedented access to resource descriptors, metadata distributed in the P2P network improves discovery of and access to resources.

To publish a new resource, the publisher specifies the publication URL via the user layer configuration. Resource descriptors are posted on this URL manually or automatically through iShare's resource publishing tool. The involved web servers serve as a repository to resource description documents and are managed

by the individual resource publishers. The P2P network in iShare consists of all participant nodes. An iShare node could be a host serving a published software service, a host published as an accessible machine resource, or the workstation from which an end user accesses iShare.

Information Naming, Publication and Discovery. Resources in iShare are hierarchically categorized into *Cyberlabs*, as shown in Figure 2. The resources in one lab are semantically related in their functionalities. Resources are described by metadata, which form a tree representing the hierarchical name space. Instead of maintaining central directory service for the tree, iShare distributes the hierarchical name space to the underlying P2P network, which supports the publication and discovery process.

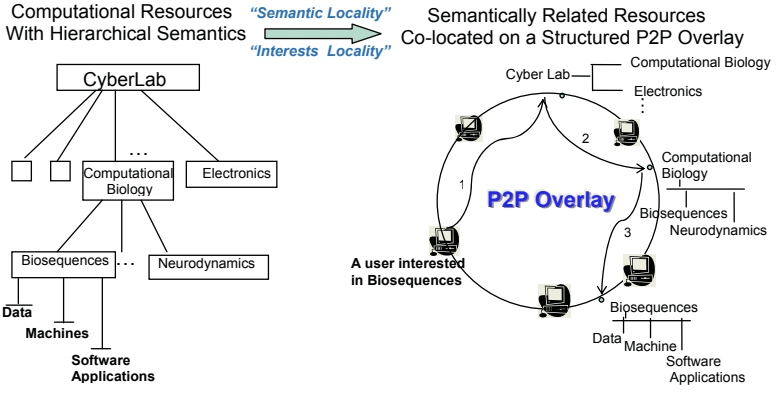


Fig. 2. Resources are organized in a hierarchical tree, which is mapped to a P2P overlay. The big circle represents a P2P overlay, with arrows indicating P2P routing messages to discover resources

Information disseminated in iShare includes resource descriptions, job execution profiles, resource-usage information, and user profiles. Each piece of information is linked to a specific resource. Thus it could be also mapped to the hierarchical structure described above. An item in the hierarchical space is mapped to a peer node by the hashing value of the item's prefix path. The current implementation of the P2P network is built on a structured overlay network, Pastry [3]. A shared data item is distilled into standardized metadata and inserted with Pastry's (*route (msg, key)*) API. Requests for data are routed without requiring any knowledge of where the corresponding data items are stored.

To achieve fault tolerance, each publication operation creates a few replicas. The discussion of the replication and fault tolerance is beyond the scope of this paper, and details could be found in [4]. A local resource cache is designed to keep recently-used metadata. Successful end-user discovery operations will update the cache. Mechanisms to maintain cache consistency are described in [4]. The

impact of caching on resource discovery latency is evaluated in Section 3. Load balancing related to the size of data stored on each peer is currently exploited and will be presented in a future paper.

2.4 Remote Resource Use

Mechanisms for the remote use of resources are at the core of most Internet-sharing systems. They include the functionality for matching discovered software services with execution platforms, employing suitable protocols for remote job execution, and connecting user input/output. iShare builds on a large number of contributions in this area. Two features distinguish iShare's remote resource use: support for roaming services and automatic creation of service portals.

Resource Allocation for Roaming Services. Roaming services are programs that pick the best combination of service replica and matching platform for every invocation. Our concepts include advanced resource matching and *open migration* of the service to the matching platform. Advanced matching needs to consider the option that services whose program source code was included in the publication may be recompiled and assigned to a different platform. Decision making for such matching in heterogeneous environments involves replication and caching strategies for service binaries and performance prediction techniques that consider possible recompilation. Open migration installs a service on a potentially unreliable platform. This involves monitoring, safeguarding mechanisms, and possibly relocating the service.

Automatic Creation of Service Portals. A service portal establishes the user interface to a remote program. It is mapped to the native job running environment at run-time. The challenges are to create this interface automatically from the service description and generate it in the user's preferred software environment. Service portals may be batch-oriented or fully interactive. They support plug-in access protocols by automatically generating job submission files and keeping track of site- and application-specific information.

3 Evaluation

Of the four areas of contributions, as described in Section 2, this section provides quantitative evaluations of the information dissemination (in terms of the latency of resource discovery) and the remote resource use techniques (in terms of the efficiency of job execution). For qualitative evaluations of the user model, and resource semantics components we refer to [5].

3.1 Efficiency of Resource Discovery

We simulated the P2P based resource discovery, *iDiscover*, on a GT-ITM router network using the transit-stub model [2]. The size of the IP network is 1050

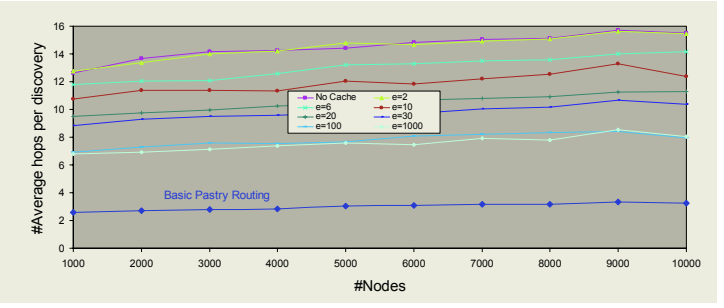


Fig. 3. Average number of hops per discovery. e is the normalized cache expiration time (cache expiration time/average request period)

routers, 50 of which are used in transit domains and the remaining 1000 in stub domains. To test the scalability of *iDiscover*, we simulated several iShare testbeds with the number of nodes ranging from 500 to 10,000. We assume the iShare nodes are uniformly attached to the routers.

In the experiment, we measured the effect of caching on discovery latency with different normalized cache expiration time e (e = cache expiration time/average request period). Each discovery operation starts from searching the root "Cyberlab". One out of five nodes in the P2P network was randomly chosen to initiate the search request. Figure 3 plots the discovery response time with e ranging from 2 to 1,000. The figure shows that the discovery latency increases very slowly with the total number of nodes. We also see that with fair expiration time ($e = 6$), the response time is reduced by 10.37% on average compared to a discovery without local cache. The measurement results for cache hit rates are analyzed in [4]. Compared with the basic Pastry message routing, the resource discovery takes only a factor of 2-4 times longer, while supporting searches of resources with specific functionalities.

3.2 Efficiency of Remote Job Execution

In this section, we compare the efficiency of remote job execution in iShare with that in SSH-based remote access. The goal is to show that iShare provides advanced network computing environment with acceptable costs in performance.

Remote access to computing services is commonly provided via explicit login to remote platforms (via REXEC, RSH and SSH). These mechanisms have several limitations [7], such as users having to manually identify and select remote machines as well as being exposed to site- and platform-specific idiosyncrasies. iShare addresses these limitations by decoupling the computing environment perceived by users from the underlying physical environment. After getting the required access (e.g., account/password, certificate or public/private keys), end users can start an arbitrary job by interacting with the service portal. iShare translates and maps the user inputs automatically to the native service interface.

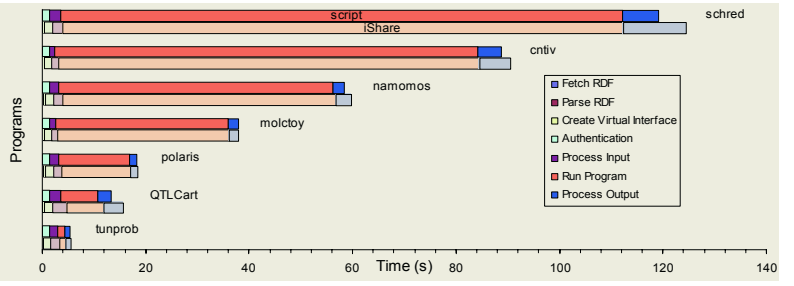


Fig. 4. Remote job execution through iShare and standard SSH

In iShare, the job execution includes the steps to fetch resource descriptor (from Web or local cache), parse the descriptor, create the service portal, authenticate, transfer input files, execute the program (including input processing and environment configuration) and process output. We chose a set of pinned services on a machine accessed via SSH. After getting accesses to these services, we ran them on iShare's built-in SSH client and monitored the time spent at each of the above steps. To test the efficiency of the standard SSH, we manually coded Shell scripts to run each of these programs on the same remote machine with standard ssh and scp commands. A ssh agent is started manually to provide the authentication similar to iShare's single sign-on mechanism (end users only need to input a password once). The efficiency difference between the public key authentication and password authentication is ignored in this experiment.

Figure 4 shows the results for running a set of programs in iShare and with the manually-coded scripts. The time for parsing RDF descriptors and creating service portals is less than one second. The SSH-based job submission in iShare performs similar to the standard SSH protocol. The standard SCP protocol outperforms the iShare's built-in file transfer protocols. The reason is that file I/O operations involved in SCP cause more overhead in iShare, because it is implemented with high level programming language (Java). Optimization of data transfer will be considered in future work.

4 Related Work

The research community is exploring a variety of approaches for constructing software infrastructures for Internet-sharing. On-going research can be divided into five categories with different foci and goals. The first category includes global standardization efforts for grid computing, represented by GGF [19] and NMI [18]. The focus of the second category is to provide application programmers a set of tools to harness "Grid" resources, e.g., to distribute massively parallel applications with message-passing. Examples of such work include Globus [8] and GridLab [11]. Work in the third category aims to develop "Grid-enabled" domain-specific applications. Active projects include EuroGrid [12],

CrossGrid [13]. Work in the fourth category is geared towards providing end users (and resource providers) with the means to disseminate, access and use networked resources. Related work in this category includes active software web portals such as PUNCH [7] and NCSA-portals [14]; and web service techniques such as IBM WebSphere Application Servers [15]. The work in the fifth category is motivated by using P2P techniques to manage and share globally distributed resources. Active research includes large-scale file sharing systems [16] and compute cycle sharing [17].

iShare belongs to the forth category, which differs from the other four categories by its user-orientation, its network accessibility to executable programs, its focus on "single-platform" rather than parallel applications and its support for generic instead of domain-specific programs. Within this category, we introduce the related work in end-user-oriented Internet-sharing systems and compare iShare in terms of the design concepts in Section 2.

User Model: Software web portals provide end users direct access to unmodified software tools via standard Web browser. However, they don't provide any open functionalities to resource providers. Most often, adding a new tool on a web portal has to involve the portal developers' administration. iShare solves this problem by decentralized web-posting and P2P message routing. Web service techniques enable users to build Web-based applications using preferred object model, programming language and platform. Service developers publish service descriptions to a central information location. This differs from iShare's support for unmodified applications with no programming requirement and decentralized resource publication.

Resource Semantics: Both web portals and web services target software application resources that are bound to fixed machine resources. iShare's roaming services combine program and machine resources provided by different communities, thus offering a more flexible and open environment for resource sharing. Resource descriptions in web portals exist as static web pages containing documentations for specific programs. These web pages are manually maintained by portal administrators. Web services use WSDL for service interface specification and the specification is registered to a central UDDI registry. The RDF language adopted in iShare focused on semantics specification rather than syntax specification in WSDL. iShare's resource description supports autonomy on the definition of access protocols, while in web portals and web services, standard access protocols are required.

Information Dissemination: Resource discovery is not a critical issue in web portals, because all application resources are listed explicitly on web pages linked through the portal's main web page. In web services, users locate the services from the UDDI registry via SOAP. In contrast to the centralized structures in the two systems, iShare disseminates information via posting on individual web pages and registering to a P2P network. There is no central registry server or storage space in the whole iShare system.

Remote Resource Use: The remote job execution in web portals and web services is supported by standardized protocols. They can be secured with HTTP basic authentication, HTTPS and SSL encryption, and digital signature. iShare supports plug-in protocols defined by individual providers via learning from the resource descriptions. The initial design of iShare supports commonly used authentication protocols such as SSH. Future work will extend iShare's security implementation to also support the Grid Security Infrastructure.

5 Conclusions

We have presented the design concepts and a implementation prototype of iShare - an Internet-sharing system built on P2P technology and the Web. iShare differs from related work in its user functionality for both providers and end users, its autonomy in defining and controlling local resources, its P2P-based information dissemination mechanisms and its support for roaming services and dynamic service portal creation. The evaluation results on resource discovery and remote execution confirm that iShare is able to deliver scalable and efficient Internet-based computing.

References

1. K. Selcuk Candan, H. Liu, R. Suvarna: Resource Description Framework: Metadata and Its Applications. ACM SIGKDD Exploration Newsletter, **3** (2001) 6–19
2. Ellen Zegura, Kenneth Calvert, Samrat Bhattacharjee: How to Model an Internetwork. Proc. IEEE INFOCOM, (1996)
3. A. Rowstron, P. Druschel: Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), (2001) 329–350
4. Xiaojuan Ren, Zhelong Pan, Rudolf Eigenmann, Y. Charlie Hu: Decentralized and Hierarchical Discovery of Software Applications in the iShare Internet Sharing System. Proc. PDCS, (2004)
5. Xiaojuan Ren, Rudolf Eigenmann: iShare – Internet Sharing of Programs, Machine and Data. Technical Report ECE-HPCLab-04203, High-Performance Computing Laboratory, Department of ECE, Purdue University, (2004)
6. Insung Park, Nirav H. Kapadia, Renato J. Figueiredo, Rudolf Eigenmann, et. al.: Towards an Integrated, Web-executable Parallel Programming Tool Environment. Proc. Supercomputing Conference, (2000)
7. Nirav H. Kapadia, Jose A. B. Fortes: PUNCH: An Architecture for Web-enabled Wide-area Network-computing. Cluster Computing, **2** (1999) 153–164
8. I. Foster, C. Kessleman: Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications, **11(2)** (1997)
9. Litzkow M. J, Livny M, Mutka M. W: Condor - A Hunter of Idle Workstations. Proc. ICDCS, (1988) 104–111
10. R. Henderson, D. Tweten: Portable Batch System: External Reference Specification. Technical Report, NASA Ames Research Center, (1996)

11. Gabrielle Allen, Kelly Davis, K. N. Dolkas, N. D. Doulamis, et. al.: Enabling Applications on the Grid - A GridLab Overview. International Journal of High Performance Computing Applications, (2003)
12. Christian Hoppe, Pallas GmbH D. Mallmann, F. Julich: EUROGRID - European Testbed for GRID Applications. GRIDSTART Technical Bulletin, (2002)
13. Marian Bubak, Maciej Malawski, Katarzyna Zajac: The CrossGrid Architecture: Applications, Tools, and Grid Services. AxGrids, (2003)
14. http://www.ncsa.uiuc.edu/AboutUs/FocusAreas/ScientificPortalsExpedition.html
15. Supporting Open Standards for Web Services and the Java Platform. ftp://ftp.software.ibm.com/software/webserver/appserv/v5/G325-1971-00.pdf
16. KazaA. http://www.kazaa.com/us/index.htm
17. D. Anderson, et. al.: Internet Computing for SETI. ASP Conference Series, 2000.
18. NFS MiddleWare Initiative. http://www.nsf-middleware.org/
19. Global Grid Forum. http://www.ggf.org/

A Service-Based Architecture for Integrating Globus 2 and Globus 3

Manuel Sánchez[1], Óscar Cánovas[2], Diego Sevilla[2], and Antonio F. Gómez-Skarmeta[1]

[1] Information Engineering and Communications Department
[2] Computer Engineering Department, University of Murcia, Spain
{msc, skarmeta}@dif.um.es
{ocanovas, dsevilla}@ditec.um.es

Abstract. During the past few years, Grid Computing has matured in terms of programming models and available tools. Some tools like the Globus Toolkit version 2 (GT2) are used in many international high performance distributed computing projects. Recently, the OGSA standard (*Open Grid Services Architecture*) has been defined, proposing a radically new philosophy compared to that of GT2. Analyzing the evolution of the scientific community working on Grid Computing, we foresee a progressive shift of current developments to this new standard, that already has a reference implementation: GT3. This paper describes the analysis and design of an architecture of OGSA Grid Services that aims to integrate both platforms seamlessly, allowing remote job invocation from GT3 to GT2 holding all the security properties, and transparent for the user.

1 Introduction and Motivation

The computing power and storage needed in scientific environments grow day by day, exceeding that offered by traditional computers. Thus, a new paradigm called *Grid Computing* emerged with the goal of sharing resources among dynamic organization coalitions in a coordinated, secure, and flexible way. Organizations belonging to the Grid can decide to share part of their resources in a controlled fashion, conforming *Virtual Organizations*[12].

Nowadays, the *Globus Toolkit*[2] is widely accepted as a *de-facto* standard for building Virtual Organizations, being GT2 the most widely used version. However, since the publication of *OGSA* (*Open Grid Services Architecture*)[11] in 2002, Globus is adapting to the emerging *Grid Service* concepts. Thus, a new version, GT3, appeared based on this paradigm, which implements all its functionality by means of *Grid Services* and standard interfaces, making the development of Grid applications easier.

In terms of security, as described in[13], the new version of *GSI* (*Grid Security Infrastructure*)[6], included in GT3, provides several advantages, such as the use of IEEE/GGF compliant proxy certificates[16], or the use of the SOAP standard

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1128–1138, 2005.

(*Simple Object Access Protocol*)[4] and the recent WS-Security specifications[3] to exchange authentication and authorization information.

Although both GT2 and GT3 try to ease the development of distributed computing applications, they have rather different mechanisms for accessing resources. While GT2 is based mainly on command line tools and scripts since it was designed for batch execution, in GT3, resources are not accessed directly, but by means of grid services, which are executed on the resources and perform a particular task.

Given the wide deployment of GT2, there are a lot of applications that rely on this Globus version. Although GT3 distributions contain an updated GT2 implementation, the migration process to GT3 does not seem to be a straightforward task: This new version involves a complete change in the operation manner which is incompatible with the previous version. Moreover, a number of organizations using both versions of Globus would want to make them interoperate. In order to achieve that integration, we have to define the elements needed to satisfy two main goals. First, the system must be able of processing the requests sent from GT3 clients, interpreting them, and building an equivalent request to be sent to the GT2 nodes. It is worth noting that this process must be transparent to the GT3 client: it should be able to use GT2 nodes as if they were OGSA grid services. Secondly, given that the request must be translated from GT3 into GT2 by, as we will show, an intermediate element that might not be trusted by both parties (for example, when the integration service is offered by a third party), it is necessary to protect the job submitted for execution to guarantee end-to-end security.

In this paper we propose an architecture based on grid services that achieve the described integration. This architecture is based on digitally signed jobs, allowing secure usage of GT2 resources by GT3 clients. It is composed, on one hand, by grid services local to GT3 clients, that is, hosted by their organization, and on the other hand, by external services hosted by third parties providing the integration service.

Similar work has been done in the GRIP project[15]. Among the main results of this project we can find some kind of interoperability between Globus and UNICORE, and the promotion of standards for interoperability in the GGF.

This paper is structured as follows. First, Section 2 provides a brief overview of the two different GT versions related to our work. Then, Section 3 describes the proposed architecture for performing the integration, that is, the different architectural elements, their relationships, and the mechanism based on digital signature that is used to protect the integrity of the submitted jobs. Section 4 shows the relevant details of the implementation. Finally, we conclude with our remarks and some future directions derived from this work.

2 Background

Nowadays, we can find several platforms for Grid Computing providing different sets of capabilities. Globus Toolkit is the most widely adopted, and is

being used in several European research projects [5, 9, 8]. In the last two years, UNICORE[10] has also emerged as an alternative toolkit for Grid Computing, as can be seen from the several existing initiatives which are trying to integrate Globus and UNICORE in a seamlessly manner[15].

2.1 Globus Toolkit 2

Globus Toolkit[2] is a computing platform composed by applications and libraries for the management, discovery and monitoring of resources. GT2 provides an uniform access interface to the computing resources, either independent nodes or a whole cluster of workstations using different operating systems. One of the main elements of GT2 is *GRAM* (*Globus Resource Allocation Manager*), which is responsible for accepting job submissions and hiding the specific details of the platform executing those jobs. In an upper layer we can also find *DUROC* (*Dynamically-Updated Request On-line Co-allocator*), a meta-manager responsible for the coordination of several GRAMs in order to execute complex tasks (composed of other jobs).

Remote execution is guided by a resource specification language (RSL). By means of RSL, users specify the job to be launched and some related execution parameters, such as the number of processors involved or required memory.

2.2 Globus Toolkit 3

GT3 represents a completely different approach from GT2, as it uses *Grid Services* as its core. Grid services are specialized *Web Services* that include some new features, such as persistence, management of notifications, and use of *service data* elements. GT3 grid services must be run in a service container, and remote invocations make use of SOAP. Grid services are self-described by means of WSDL (*Web Service Description Language*) documents, which must be obtained by clients in order to access them. Complexity is reduced by means of client and server stubs, which are intermediate software elements derived automatically from the WSDL description. Therefore, details about SOAP and other technological elements are hidden from the developer's point of view.

Two of the most important novelties in GT3 are service data and notifications. The former allow the programmer to add *structured data* to the services, which can be accessed through a well defined interface. Then, service data can be used to expose the grid service internal state or metadata. Notifications are used by a grid service or *notification source* to notify changes in its state to any subscribed client or *notification sink*. Notifications are closely related to service data, because a client is not subscribed to a whole grid service, but to a specific service data belonging to the service.

The Java CoG Kit[1], in turn, offer a Java framework for accessing GT2 programs and services. In general, Commodity Grid (CoG) kits allow users, developpers and administrators to access the grid from a higher-level framework.

2.3 Security Mechanisms for Globus: GSI

Both GT2 and GT3 use the security services provided by GSI (*Globus Security Infrastructure*)[6]. GSI makes use of X.509 certificates for authentication purposes, and TLS for establishing confidential channels. It also supports different authorization policies, delegation of privileges, and can also be integrated with other local security systems being used in the organization.

3 Proposed Architecture

3.1 Requirements for the Integration of GT2 and GT3

The integration of both versions of Globus has a set of requirements derived from the intrinsic internal architectural organization of both platforms. First, intermediate elements with support for both GT2 and GT3 are necessary. Second, a RSL request must be built that describes the job execution request made by the user. Also, the code, input data, and output data must be transmitted. It would also be desirable that the client, the intermediate element, and the GT2 nodes in charge of the final execution could be located in different administrative domains, thus not having to belong to the same organization. Shielding the user of the details of GT2 (such as the RSL specification or the use of the GRAM protocol) is also desirable. Finally, the semantics of the execution stated by the GT3 client must be preserved along all the process, that is, the architecture must guarantee that neither the code nor the input or output data has been forged, as well as that the identity of the caller is not supplanted by any other entity.

3.2 The Elements of the Proposed Architecture

We have proposed a generic architecture with three different administrative domains, as shown in Figure 1. First, the domain of the GT3 client that wants to execute a job in a GT2 cluster. For the sake of simplicity, we suppose that this domain is composed exclusively of GT3 nodes, possibly connected to other GT3 nodes on different sites conforming a Virtual Organization. Second, an intermediate administrative domain exists to host the integration service, thus having to support both versions of Globus. This intermediate element can be seen as an enterprise (organization) dedicated to offer interconnection among Grids. The last administrative domain in the picture is the destination one, in which the execution of the job will be performed, composed of one or more GT2 nodes.

The main elements participating in the integration are the following:

- **GT2Gateway Service.** To avoid GT3 users having to explicitly build the GT2 RSL request, an OGSA service is introduced that builds this specification from the data given by the user (executable file, parameters, data files, etc.) This service is also responsible for signing the RSL and all the files on behalf of the user and for verifying the signed output generated by the GT2 nodes, in order to guarantee end-to-end integrity and authentication. The

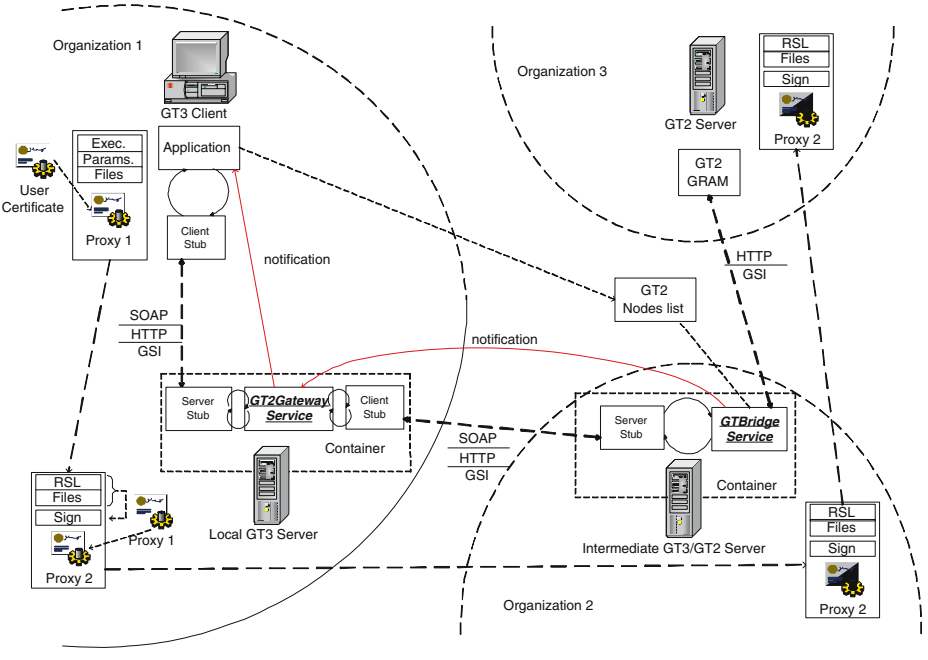


Fig. 1. Architecture of the proposed solution

GT2Gateway service must be running in the local domain to which the user belongs, because this service has to build and sign the whole description of the job. Besides, this service acts as a notification source for the client. We can find a similar approach to provide job integrity by means of job signing in UNICORE[10].

- **GTBridge Service.** This service offers two main functions: First, it is responsible for maintaining a list of GT2 nodes (or set of nodes) that are available to be used from GT3. This way, the user should first check this list (for example, using the *Index Service*) to see what nodes are available before interacting with the *GT2Gateway* service. And second, once the RSL is built, it receives from the *GT2Gateway* service the Job description, input files and executables, and the certificates needed to verify the signature, as we will see later. From this point on, the request has been converted into the GT2 format, and therefore it can be submitted to a GT2 GRAM server of other administrative domain. After executing the job, the results are collected by this service, and put in knowledge of the client.
- **GT2 GRAM with digital signature support.** The functionality of the GT2 GRAM server must be augmented (in form of a plug-in) in such a way that allows us to interpret digitally signed job descriptions. For that purpose, the RSL must indicate that this particular job is protected using a digital signature, making it to be interpreted by the modified GRAM. It

is worth noting that this extension can be added without changing the base Globus installation in the node. Once the job is executed, the results are also protected against integrity attacks by means of a digital signature.

Finally, we also have to assure end-to-end security. Although GSI guarantees a secure communication between the entities in homogeneous environments, the introduction of an intermediate entity (the integration service) makes it necessary to deeply analyze the security implications of the integration.

3.3 The Integrity Problem

The proposed solution could have the problem of having to trust an external entity (the mediator) to which the execution of jobs is delegated. This external entity could modify the jobs before sending them for execution or even impersonate the user sending out jobs for execution without the user's knowledge. To overcome this problem, the job description is signed with the user's private key. This way the destination GT2 node can check the authenticity of the data received. Therefore, management of proxy certificates and private keys involved in the process is of paramount importance.

The Globus Security Infrastructure is based on the use of restricted user proxy certificates. These certificates are issued by the user for a temporal user-controlled key pair, and can be used to delegate operations to other elements of the system in a secure and controlled way (those delegation chains are not bounded). As can be seen in Figure 1, the GT3 client first generates a proxy that will be used by the *GT2Gateway* service to sign the job description. After that, this service generates another proxy, that will be sent, jointly with the signed job, to the *GTBridge* service. This second proxy, together with its certificate chain, will be used by the *GTBridge* service to contact the GT2 node and execute the job on behalf of the user. As the job description is signed with the first proxy, which is not accessible by the intermediate service, the contents of the job cannot be modified. Moreover, the use of the second proxy will allow executing the job on behalf of the user, thus applying the security policies mapped to that specific user (known as "*grid mapping*" in Globus). Also note that the architecture of GT3 imposes that a proxy certificate must be created for each service invocation.

4 Implementation of the Proposed Architecture

Once we have analyzed the main elements of our architecture, and having in mind the requirements imposed by such approach, we provide some details related to the implementation and operation of those elements.

4.1 Operation Steps

Figure 2 shows the main steps involved in the execution of GT2 jobs that have been submitted from a GT3 client.

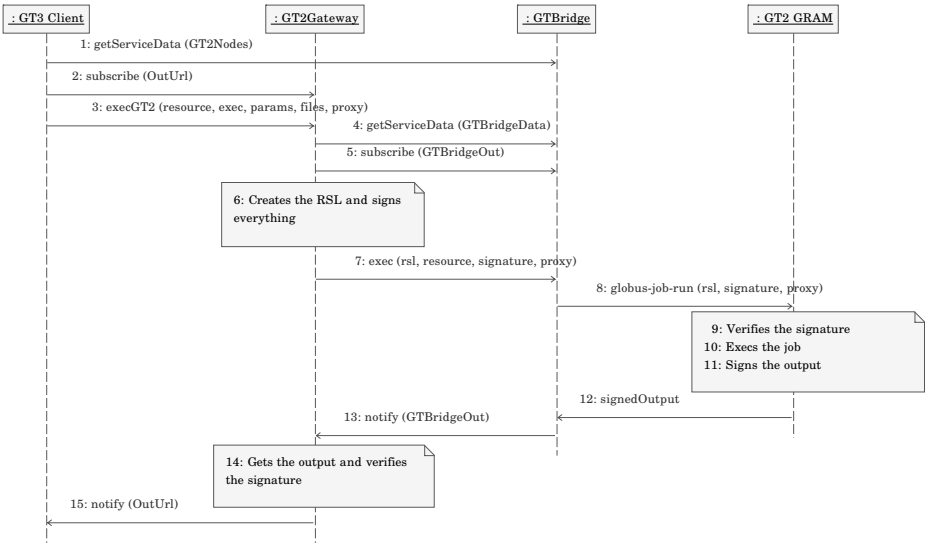


Fig. 2. System interaction

1. A client obtains the information about the available GT2 nodes from the service data elements managed by *GTBridge*. Optionally, the client can delegate to *GT2Gateway* the selection of a specific set of nodes.
2. The client subscribes to the *OutUrl* notification of the *GT2Gateway* service to be notified about how to get the results when the job has been executed.
3. After the selection of nodes, the client sends to the *GT2Gateway* the name of the program, any input files or parameters, an identifier of the selected GT2 node and a proxy certificate generated from the user certificate.
4. *GT2Gateway* gets the URL where *GTBridge* will leave the signed job from the *GTBridgeData* service data to add it to the RSL which it is building.
5. Then, this service subscribes to the *GTBridgeOut* notification of the *GTBridge* related to the availability of results derived from the job execution.
6. *GT2Gateway* builds the RSL document that describes the job, including a random identifier which will be used by the GT2 node as a reference for the client session. On the other hand, *GT2Gateway* creates a second proxy from the proxy certificate submitted by the user. Finally, the RSL job description, and its related input files, are digitally signed by *GT2Gateway* making use of the private key associated to the first proxy certificate.
7. *GT2Gateway* uses the *GTBridge* service to request the remote execution in the GT2 node, providing the RSL description, its related files, the digital signature, and the new proxy certificate.
8. *GTBridge* uses the interface provided by the Java CoG Kit to submit the job to the target GT2 node.
9. The target GT2 node checks whether it is processing a signed job (which is specified in the RSL description in order to differentiate unsigned jobs

submitted by other GT2 nodes from GT3-originated jobs). It also verifies that the RSL description is properly signed using the first user proxy. Furthermore, the proxy used for authentication purposes must also be the last element of the trusted chain of certificates presented by the user.
10. Using the grid mapping policy, the user is mapped into a local user, and the job is executed according to that user's constraints.
11. Once the job has finished its execution, the GT2 node digitally signs the standard output and any other output files derived from the job execution with the host private key.
12. Next, the GRAM server sends the signed execution output to the *GTBridge*.
13. *GTBridge* specifies the URL where the signed output is available in the appropriate service data, triggering the notification to *GT2Gateway* Service.
14. Finally, *GT2Gateway* makes use of those service data elements in order to obtain the different outputs.
15. The output will be considered valid after verifying that the related digital signature is valid and that the signer certificate belongs to the GT2 Node. In that case, the GT3 client will be notified about the availability of results.

As we can see from these steps, clients are not aware of any operation related to RSL descriptions or digital signatures. The main goal is to achieve an integration as seamless as possible, which might be replaced by a different implementation in a transparent manner.

4.2 Some Details About the GT3 Services

Our OGSA services are specified using GWSDL, including definitions of some of the different service data elements used for notification purposes. Those services implement some OGSA standard interfaces, such as `NotificationSource` or `Factory`, in order to deal with each request in an independent and persistent manner. Figure 3 shows part of the GWSDL of *GT2Gateway*.

Digital signatures follow the *PKCS#7*[14] standard, since this type of document contains information about the signature, the data being protected, and the certificates composing the verification chain.

4.3 Some Details About the Plug-in for the GT2 GRAM Server

The GT2 GRAM server should be extended for three reasons. First, it must be able to understand the new RSL attributes for digital signature support. Second, it has to verify the different user proxies and digital signatures. Finally, it must deal with the different results derived from job executions, also signing the different outputs in order to protect them from an external modification or forgery. Everything has been implemented as a plug-in for the *Job manager*.

To do this, when the GRAM receives a new job, it checks if it is a signed job. In this case, first of all, the GRAM gets the location (URL) of the signed file from the received RSL document. Then, it makes use of the Globus GASS API

```
<types>
    <xsd:element name="runGT2OutputMessage">
       <xsd:complexType/>
    </xsd:element>
    <xsd:element name="runGT2InputMessage">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="exec" type="xsd:string"/>
                <xsd:element name="params" type="tns:stringArray"/>
                <xsd:element name="resource" type="xsd:string"/>
                <xsd:element name="files" type="tns:stringArray"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</types>

<gwsdl:portType name="GT2GPortType"
      extends="ogsi:GridService ogsi:NotificationSource">
    <operation name="runGT2">
        <input message="tns:runGT2InputMessage"/>
        <output message="tns:runGT2OutputMessage"/>
        <fault name="Fault" message="ogsi:FaultMessage"/>
    </operation>
    <sd:serviceData name="GT2Nodes"/>
    <sd:serviceData name="OutUrl"/>
</gwsdl:portType>
```

Fig. 3. `GT2Gateway` service definition.

to fetch this file. Once this is done, the digital signature, the certification chain and the RSL can be verified. Finally, when the job has been executed, the files generated in the execution are signed using the host private key, and included in a new PKCS#7 document. This document is transferred to the *GTBridge* Service using again the Globus GASS API.

4.4 State of the Implementation and Tests

We have tested the architecture allowing users to execute a file compressor using the GT2/GT3 integration service. A GUI to submit jobs to the GT2 node has also been developed, so that the user can specify the GT2 node to send the job, the needed files, and the command to be executed in the target node. Once the execution ends, the user is notified and can get the generated files using the same tool. This test bench is a starting point to prove the feasibility of the proposal, as we intend to extend it to support the new WSRF specification.

5 Conclusions and Future Directions

In this paper we outline the need for a real integration of GT3 and GT2 nodes in order to achieve a progressive shift of current developments to the OGSA framework in a seamlessly manner. We propose an architecture based on intermediate services which make use of digital signature mechanisms and proxy certificates to provide integrity and authentication security services.

We are currently working on the extension of our architecture by adding new operations related to the life-cycle of jobs, such as monitoring, migration, etc.

Furthermore we are currently implementing an automated mechanism for smart selection of GT2 nodes guided by some parameters such as computational load, network bandwidth or economic costs.

Although the new Globus Toolkit version (GT4), based on the new WSRF[7] specification, is in an advanced state of development, our work can be easily adapted to that version, as both GT3 and GT4 are conceptually equivalent, except for some changes of syntax and terminology.

References

1. CoG Kits home page. http://www.cogkit.org.
2. Globus toolkit home page. http://www.globus.org.
3. B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, C. Kaler, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon. *Web Services Security (WS-Security). Version 1.0*, 2002.
4. Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. *Simple Object Access Protocol (SOAP) 1.1*, 2000.
5. Marian Buback, Jesus Marco, Holger Marten, Norbert Meyer, Marian Noga, Peter A.M. Sloot, and Michal Turala. CROSSGRID - Development of grid environment for interactive applications, 2002.
6. R. Butler, V. Welch, D. Engert, I. Foster, S. Tuecke, J. Volmer, and C. Kesselman. A national-scale authentication infrastructure. *IEEE Computer*, pages 60–66, 2000.
7. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & evolution, 2004.
8. F. Donno, V. Ciaschini, D. Rebatto, L. Vaccarossa, and M. Verlatto. The WorldGrid transatlantic testbed: a successful example of Grid interoperability across EU and U.S. domains. In *Proceedings of the Conference for Computing in High Energy and Nuclear Physics*, 2003.
9. F. Donno, L. Gaido, A. Ghiselli, F. Prelz, and M. Sgaravatto. DataGrid prototype 1. EU-DataGrid collaboration. In *Proceedings of TERENA Networking Conference*, 2002.
10. D. Erwing, H. Ch. Hoppe, S. Wesner, M. Romberg, P. Weber, E. Krenzien, P. Lindner, A. Streit, H. Richter, H. Stuben, V. Huber, S. Haubold, and E. Gabriel. *UNICORE Plus Final Report*, 2003.
11. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. *The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems integration*, 2002. Draft.
12. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid. enabling scalable virtual organizations. *Supercomputer Applications*, 2001.
13. J. Gawor, S. Meder, F. Siebenlist, and V. Welch. *GT3 Grid Security Infrastructure Overview*, 2003. Draft.

14. RSA Laboratories. *PKCS#7: Cryptographic Message Syntax, Version 1.5*, 1993. An RSA Laboratories Technical Note.
15. M. Rambadt and P. Wieder. UNICORE - Globus: Interoperability of Grid infrastructures. In *Proceedings of Cray User Group*, 2002.
16. S. Tuecke, D. Engert, I. Foster, V. Welch, U. Chicago, M. Thompson, L. Pearlman, and C. Kesselman. *Internet X.509 Public Key Infrastructure Proxy Certificate Profile*, 2003. Internet Draft.

The CampusGrid Test Bed at Forschungszentrum Karlsruhe

Frank Schmitz and Olaf Schneider

Forschungszentrum Karlsruhe, Institut of Scientic Computing (IWR),
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen

Abstract. A central idea of Grid Computing is the virtualization of heterogeneous resources. To meet this challenge the Institute for Scientific Computing (IWR), has started the project CampusGrid. Its medium term goal is to provide a seamless IT environment supporting the on-site research activities in Physics, Bioinformatics, Nanotechnology and Meteorology. The environment will include all kinds of HPC resources: vector computers, shared memory SMP servers and clusters of commodity components, InfiniBand-Clusters as well as a shared high-performance SAN storage solution and a global file system. The paper shows the ideas, the test-bed and informs about the current project status and scheduled development tasks. This is associated with reports on other activities in the fields of Grid computing and high performance computing at IWR and D-Grid.

1 Introduction

It is a medium term goal to realize the project CampusGrid at FZK to support the on-site research activities in physics, bioinformatics, nanotechnology and meteorology. We cooperate with industrial partners and international institutions to test and discuss the functionality, robustness and scalability of such a seamless IT environment. The CampusGrid project aims at building a virtual computer with

- CPUs of various types, running different operating systems
- distributed and shared memory
- a global high performance file system

The environment will be based on a middleware layer which uses Grid technologies to implement secure single sign-on, job management and allocation of resources. This middleware layer may be assembled from the wide range of existing or currently developed Grid middleware solutions which will be adapted and improved in order to match our specific requirements. For instance, the middleware needed on each resource of the CampusGrid must support our heterogeneous hardware and should be as light-weight as possible. For compatibility, all resources of the CampusGrid should be accessible as OGSA compliant Grid services. Our close cooperation with international Grid projects as well as industrial partners – hardware manufacturers and software developers – guarantees a progressive and competitive solution.

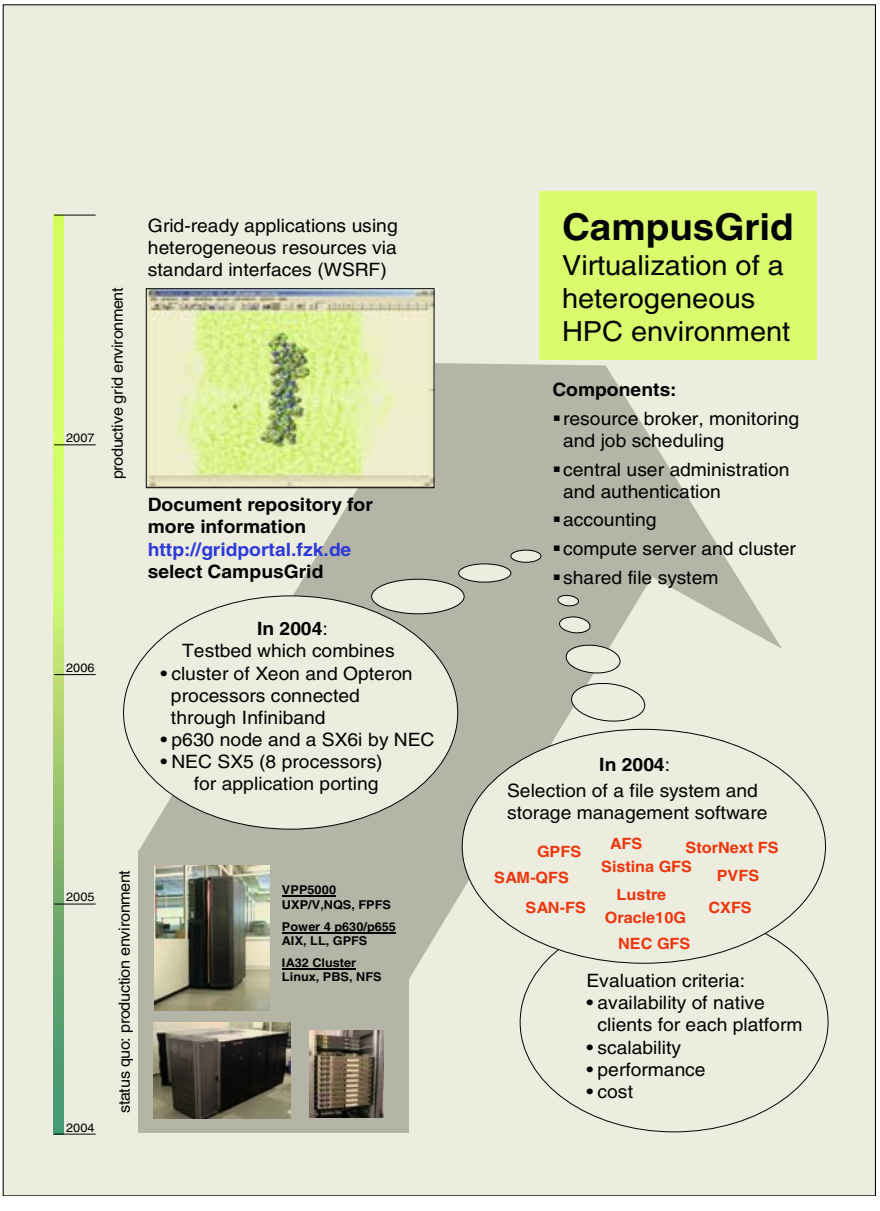
P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1139 – 1142, 2005.

Modern application packages often include parts well suited for cluster computing as well as serial components which require vector processors to achieve acceptable performance. The processing of experimental data typically results in a complex computational scheme where the user divides the job into subtasks and submits them to the available computing resources. By use of an intelligent brokerage and scheduling mechanism the CampusGrid software will choose the optimum computing platform for each subtask with respect to resource workload and user constraints. This will allow for the complementary use of high performance vector computers, large SMP systems and clusters built from commodity hardware (with or without high performance interconnect) in the same job. At the same time emphasis is put on the implementation of a unique working area with high performance access for all CampusGrid users. Access to online storage in an effective way and improved communication between hundreds or thousands of nodes in High Throughput Computing (HTC) clusters will benefit from technological standards like iSCSI and Infiniband. Both options will be studied in order to optimize the dataflow between the CampusGrid components. The results of these studies are of special interest also for the rapidly scaling storage area networks of the GridKa environment.

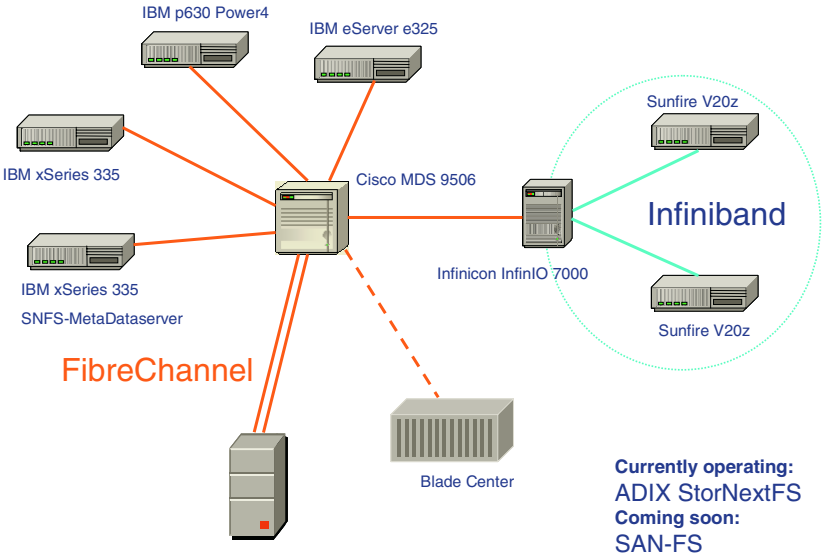
It is the overall long term vision to allow the seamless integration of all kind of resources in a Grid, CPUs and data archives as well as on-site experiments, measuring devices and facilities like the synchrotron radiation source ANKA (www.fzk.de/anka). Efficient use of such a computational environment will also require adaptations of the application software which will be demonstrated by selected packages of scientific and industrial partners who evaluate the CampusGrid under production conditions.

2 Status Quo and Project Progress

For many years, IWR has been operating a wide range of different computers which are optimally tailored to their respective use, like vector computers (VPP5000), clusters of SMP computers (IBM-SP Power3/Power4) and clusters of commodity processors. In its responsibility for the CampusGrid Project, IWR plans to harmonize heterogeneous hardware platforms by a software layer starting in 2004. For this purpose, the concept of grid computing will be applied to the existing installations. A mandatory first step is the administration of the users and access rights by a common user administration in the Windows based "active directory with services for Unix". Anther step is the implementation of global data storage, i.e. all data inventories may be accessed and changed in the same way by each computer connected. As an ad hoc solution the corresponding environment “Globale Datenhaltung und Benutzerverwaltung” has already been deployed. Now we are in the process of evaluating the hard- and software environment for the CampusGrid project.


CampusGrid
Virtualization of a heterogeneous HPC environment
Components:
resource broker, monitoring and job scheduling
central user administration and authentication
accounting
compute server and cluster
shared file system
Grid-ready applications using heterogeneous resources via standard interfaces (WSRF)
productive grid environment
2007
Document repository for more information
http://gridportal.fzk.de
select CampusGrid
In 2004:
Testbed which combines
• cluster of Xeon and Opteron processors connected through Infiniband
• p630 node and a SX6i by NEC
• NEC SX5 (8 processors) for application porting
2006
In 2004:
Selection of a file system and storage management software
GPFS
AFS
StorNext FS
SAM-QFS
Sistina GFS
PVFS
SAN-FS
Lustre
Oracle10G
CXFS
NEC GFS
Evaluation criteria:
• availability of native clients for each platform
• scalability
• performance
• cost
2005
VPP5000
UXP/V,NQS, FPFS
Power 4 p630/p655
AIX, LL, GPFS
IA32 Cluster
Linux, PBS, NFS
status quo: production environment
2004

The active Testbed for SNFS looks like:



A Model for Flexible Service Use and Secure Resource Management

Ken'ichi Takahashi[1], Satoshi Amamiya[2], and Makoto Amamiya[2]

[1] Institute of Systems & Information Technologies/KYUSHU,
2-1-22 Momochihama, Sawara-ku, Fukuoka, 814-0001, Japan
takahashi@isit.or.jp
[2] Faculty of Information Science and Electrical Engineering, Kyushu University,
6-1 Kasuga-Koen, Kasuga-shi, Fukuoka 816-8580, Japan
{roger, amamiya}@al.is.kyushu-u.ac.jp

Abstract. Grid computing is promissing as an infrastructure that allows users to use distributed computer resources by simple connecting a computer to the network without special operation; just like connecting to the electricity, water or gas grid. In this paper, regarding resources as services, we propose a new architecture for realizing an environment in which users can use services that are in various locations through their portable terminals. In this architecture, a service is managed by an *agent*, which has two resource management spaces named the *Public Zone* and the *Private Zone*. The Public Zone is a space for realizing flexible public service use. The Private Zone is a space for protecting private user information. Moreover, agents are organized in a group called the *community* and are managed independently in each community. Thus, we realize both of flexible service use while protecting private information.

1 Introduction

Grid computing is promissing as an infrastructure that allows users to use dispersed computer resources by simply connecting a computer to the network with no special operation; just like connecting to the electricity, water or gas grid. SETI@home[5] and distributed.net[1] are the two well-known grid computing projects. These projects try to search for extraterrestrial intelligence or carry out cryptographic analysis by using CPU resources connected to the Internet. In these projects, a resource is a CPU resource. But a resource is not only a CPU resource, but also data and a service. If we regard a resource as a service, we will be able to realize an environment that allows users to use services provided in various locations through their portable terminals. For example, if a user is in the laboratory, he can use the printer and the copy machine in the laboratory through his portable terminal; if he is in his house, he can use the television, and the audio player there and so on. In this way, users will be able to use services which are based on their locations. To realize such an environment, the following functions are required.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1143–1153, 2005.

Service-Use Mechanism. Each service has a method for its use. For example, when we use a telephone, first, we pick up the telephone receiver and put in coins and dial. In a same way, we must get the method for using a service and use the service according to the method appropriate to it.

Protection of Private Resources. In the real world, various services are provided in exchange for resources/information like money and e-mail address. So, when a user receives a service, he may have to provide some private information. But users don't want to unconditionally make their own information public. Therefore, it is necessary to protect their resources/information.

Decentralized Service Management. In this environment, there are countless services. So it is difficult to manage and dispatch resources to users in a centralized way. Therefore, we need a mechanism for managing resources in a group depending on the location and/or other indicators.

The availability of a service-use mechanism depends on the degree of protection of private resources. If a user does not provide all his information, the services he can use are limited; if he provides more information, he may be able to use more services. Therefore, we need to balance these two functions. In this paper, we propose a new architecture based on two agent systems, named KODAMA[6] and VPC[3]. In this architecture, a service is managed by its *agent*, which has two resource management spaces named the *Public Zone* and the *Private Zone*.

The Public Zone is a space for realizing flexible public service use. The Private Zone is a space for protecting private user information. Moreover, agents are organized in a group called the *community* and are managed independently in each community. Thus, we realize the flexible service use and the protection of private information.

2 The Public Zone and the Private Zone

In this section, we introduce the Public Zone and the Private Zone. In our architecture, agents have a Public Zone and a Private Zone. The Public Zone is for flexible public service use. The Private Zone is for the protection of private resources. An overview of our architecture is shown in Fig. 1.

The Public Zone manages *public resource*s. A public resource is a resource, like a service or information. Public resources are open to the public. A public resource has a *public policy* which consists of a method for its service use and a specification of attributes. A user agent acquires a public policy from the service provider agent and uses its service by behaving according to its method.

A *Security Barrier* exists between the Public Zone and the Private Zone. The Security Barrier has functions for restricting access to private resources and for restricting communications of a program which accesses private resources.

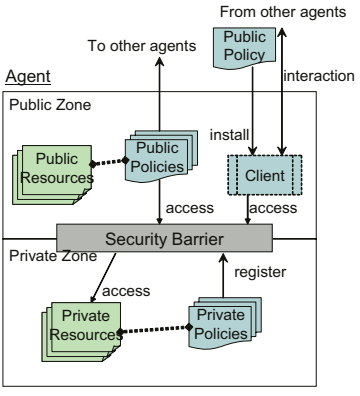


Fig. 1. An Overview of the Public and Private Zone Model

The Private Zone manages *private resources*. An agent cannot directly access private resources, but must access them through the Public Zone. A private resource has a *private policy* which consists of a method for accessing its resources and attributes governing access permission. Private policies are registered with the Security Barrier. The Security Barrier monitors access to private resources and communications of a program which has accessed private resources.

2.1 Public Policies and Private Policies

Each resource has a public policy or a private policy both of which consist of a method and attributes. A method is implemented by a program called the *client program*. The details of the client program are introduced at Sect. 2.3. Attributes consist of common attributes and characteristic attributes.

The common attributes are composed of *owner*, *type*, *description* and *agent_attr*. The owner attribute is the name of the agent which generated the policy. The type attribute shows whether the policy is a public policy or a private policy. The description attribute gives an explanation of the resource. The agent_attr is the list of attributes of the agent which managed the policy.

The characteristic attributes of a public policy are *dependency* and *communication*. The dependency attribute shows what resources are needed for using the resource. The dependency attribute is defined as the list of description attributes of the public policy. When an agent wants to use its resource, the agent must gather resources specified in the dependency attributes in advance. The communication attribute shows whom it is necessary to communicate with for using the resource. The value of the communication attribute can be any one of *no_communication*, *owner_only* or *agent_list*. No_communication means that no communication is required. Owner_only means that it requires only communications with the resource provider. Agent_list is defined as the list of agent name and means that it requires only communications with agents specified in its list.

The characteristic attributes of a private policy are *access* and *allowable_communication*. The access attribute specifies attributes of programs which are permitted the access the private resource. The allowable_communication attribute specifies communications allowed to the client program which accesses the private resource. The value of the communication attribute can be any one of *allow_all*, *agent_list*, *owner_only* and *deny_communication*. Allow_all permits only communications with agents specified in the communication attribute of the client program. Agent_list permits only communications with agent specified in its list. Owner_only permits only communications with the distribution origin (represented by the owner attribute) of the client program. Deny_communication denies all communication.

2.2 The Security Barrier

The Security Barrier is prepared for the protection of private resources between the Public Zone and the Private Zone in each agent. All the access must be done through the Security Barrier. The Security Barrier forbids the access from other agents and also checks the access from programs in the Public Zone. In this architecture, each private resource has a private policy. The private policy is registered with the Security Barrier. The Security Barrier protects private resources by restricting the access to private resources and restricting communications of the client program.

When the client program accesses to a private resource, the access is checked by the Security Barrier. The Security Barrier compares the common attribute (owner, description and agent_attr) of the client program and the access attributes of the private resource. If the access is accepted, the Security Barrier returns its value and registers the client program with the access-list; if it is rejected, IllegalAccessException happens. After that, the Security Barrier monitors the communication of the client program in the access-list. When the client program communicates with other agent, the Security Barrier compares the communication partner and the allowable_communication attributes of the private policy. Then, if its communication is allowed, the client program can communicate; if it is rejected, IllegalAccessException happens. In this way, the Security Barrier protects private resources by restricting the access and the communication.

2.3 The Client and the Service Program

In this architecture, each service is provided by one agent. A user agent uses a service by communicating with the service provider agent according to the method for use of the service. But methods for service use are different for each service. Therefore, it is difficult to implement an agent which is able to ab initio use various services. Therefore, we define the *service program* and the *client program* as a pair.

A service provider agent sets up a service program and a client program (as a part of the public policy) in its Public Zone. A user agent acquires a client program from a service provider agent and invokes it in its Public Zone. Then,

Table 1. Additional Methods for the Service and the Client Program

Result accessToPublicResource(*service_desc*) Call a client/service program specified in *service_desc* on the Public Zone and return its result
Result accessToPrivateResource(*service_desc*) throws IllegalAccessException Call a client/service program specified in *service_desc* on the Private Zone and return its result
void inform(*agt_name*, *method*, *par*, *res_method*) throws IllegalAccessException Send an invocation of the *method(par)* to *agt_name*. When the agent receives a response, it invokes the *res_method(response)*
Result accessTo(*agt_name*, *method*, *par*) throws IllegalAccessException Send an invocation of the *method(par)* to *agt_name* and suspend the program. When the agent receives a response, the program works again
void reply(Result *res*) Send *res* back

the service is actualized by communications, guided by the client program and the service program, between the service provider and user. The service and client program are implemented in Java with additional methods as shown in Table 1.

3 Decentralized Service Management

In an environment with a lot of services, it is difficult to manage and dispatch resources to users in a centralized way. Therefore, a decentralized service management mechanism is required. For that, we define the *community*. Each community manages agents in its community independently. Each community has a *portal* agent which is the representative of each community. A portal agent has *tax policies* that are obligations imposed on agents in its community. Agents are able to use services provided in the community to the extent that they fulfill their obligation.

3.1 The Tax Policy

The tax policy is the obligation imposed on agents who have joined a community. The tax policy is designed for checking the qualification to join the community and/or for obligating to provide the service. The tax policy consists of a client program and attributes. The attributes are the same as for the public policy. When agents join a community, they must install the client program specified in the tax policy in their Public Zone.

When an agent joins a community, it receives tax policies from the portal agent of the community. The agent installs the client programs in its own Public Zone and notifies that to the portal agent. If the community needs to check

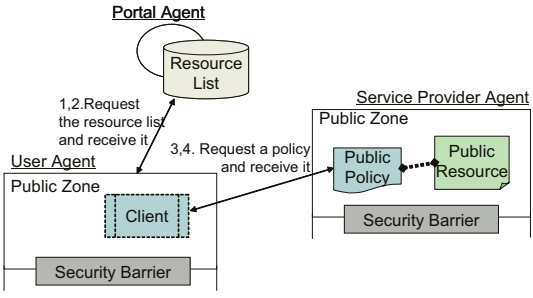


Fig. 2. The Steps for Obtaining a Client Program

the qualification to join the community, the portal agent accesses to the client program in the agent's Public Zone. Then, the client program accesses resources of the agent for checking the qualification to join the community[3] and returns its result. As the result, the portal agent sends a participation permission/refusal notification message. If the agent receives a participation permission notification message, the agent joins the community. Moreover, agents in the community provide the services specified in the tax policies to other agents in the community.

3.2 The Registration of the Service

The portal agent manages resources provided by agents in the community. When an agent joins the community, it sends owner and description attributes to the portal agent. The portal agent registers them in the resource table, allowing an agent to find resources by querying to the portal agent.

3.3 Service Use

An agent acquires a client program from a service provider agent and uses a service. The steps for obtaining a client program are shown in Fig. 2.

1. A user agent sends a resource list request message to a portal agent.
2. The portal agent returns the resource list (which consists of owner and description attributes).
3. The user agent finds necessary resources from the resource list.
4. The user agent requests a public policy from the service provider agent (indicated by the owner attribute in the resource list).
5. The service provider agent returns the requested public policy.
6. The user agent installs the client program detailed in the received public policy.

[3] A method for checking the qualification is not shown in this paper, because it depends on applications.

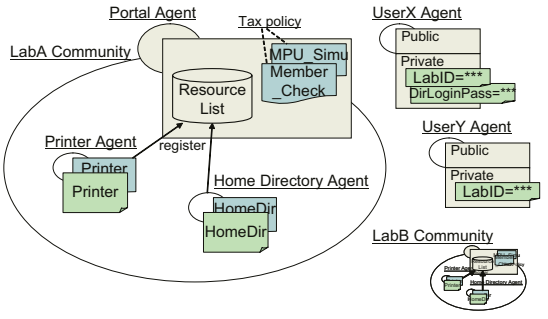


Fig. 3. The Application Overview

In this way, an agent acquires a client program. Subsequently, the user agent acquires public policies indicated in the dependency attribute of the received public policy. Finally, the user agent invokes the client program.

4 An Application Example

In this section, we show an application example in which a member of a laboratory makes use of services provided in the laboratory. In the example, we assume that sensors for detecting user's location are installed in the laboratory and notify the user agent of the community name corresponding to each user's location. We also assume that the problems of tapping, spoofing and masquerading have been solved by importing technologies of encryption, digital signature, Public Key Infrastructure and so on.

4.1 The System Structure

The application overview is shown in Fig. 3.

In this application, there is a printer agent and a home directory agent in the LabA community and the LabB community. A printer agent provides a printer services. A home directory agent provides home directory access services for users who have a login password. Also, LabA is simulating MPU (MicroProcessor Unit) processes that requires more CPU power. Accordingly, the portal agent of the LabA community has a tax policy (*MPU_Sim*) which supplies the CPU resources for the simulation, and a tax policy (*Member_Check*) which confirms whether agents are members of its laboratory or not.

UserX/userY are members of LabA/B, repectively. They have a *LabID* with the following attributes:

```
owner="UserX/Y", type="private", agent_attr=attribute of userX/Y,
description="Belonging to laboratory",
```

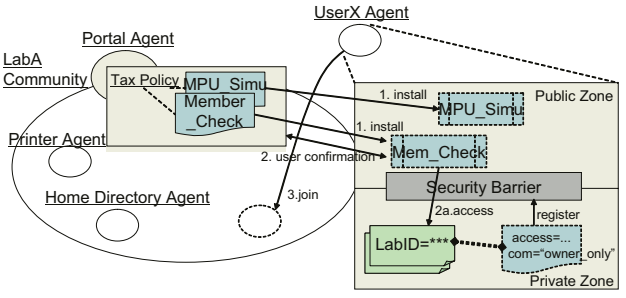


Fig. 4. The Behavior when the UserX agent Enters the LabA community

```
access=agent_attr:{owner="portal of LabA/LabB"},
allowable_communication="owner_only"
```

. Also, userX has a *HomeDirLoginPass* with the following attributes:

```
owner="UserX", type="private", agent_attr=attributed of userX,
description="Home directory login password",
access=agent_attr:{owner="Home Directory Agent"},
allowable_communication="owner_only".
```

4.2 Behavior When Entering the Laboratory

The behavior when userX enters LabA is shown in Fig. 4. When userX visits LabA, its community name (LabA) is notified to his agent by sensors. The agent then receives tax policies (MPU_Simu, Member_Check) from the portal agent of the LabA community and installs their client programs in its own Public Zone; the portal agent accesses Member_Check installed in userX's Public Zone and tries to confirm whether he is a member of LabA. The Member_Check program tries to access the LabID. Then, the access is allowed because attributes of LabID are `access=agent_attr:{owner="portal of LabA"}`, `allowable_communication="owner_only"`. As the result, the confirmation succeeds. After that, the userX agent receives a participation permission notice message and joins the LabA community. On the other hand, even if userY tries to join the LabA community, he can not join because attributes of his LabID are `Access=agent_attr:{owner="portal of LabB"}`.

And agents in the LabA community have the MPU_Simu program in their Public Zone, because MPU_Simu is specified in the tax policy of the LabA community. Therefore, agents in the LabA community supply the CPU resources for the MPU simulation. In this way, we can simulate MPU processes using CPU resources of other agents through their MPU_Simu program.

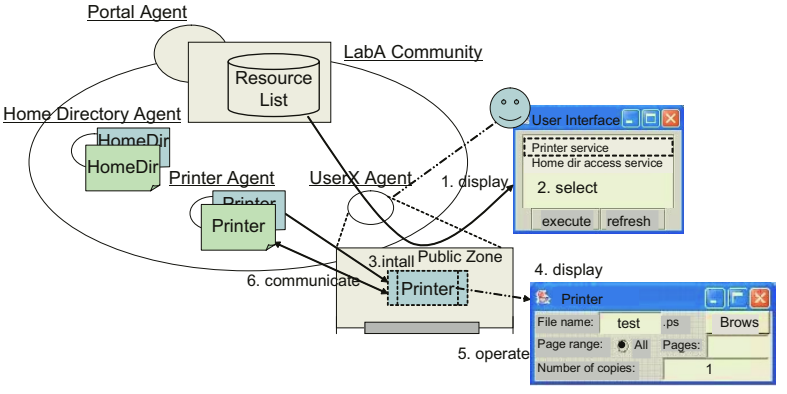


Fig. 5. The Behavior when a User Agent Uses a Service

4.3 Service Use in the LabA Community

The behavior involved in using a service in the LabA community is shown in Fig. 5. A user agent receives the resource list. The user agent shows its description attributes to the user. If the user selects a service he wishes to use, the user agent acquires the public policy and invokes the appropriate client program.

For example, when userX enters LabA, "Printer service" and "Home dir access service" are shown on his portable terminal. If he selects "Printer service", his agent acquires the Printer policy and invokes its client program. This results in the interface for "Printer service" being shown on his portable terminal, and he uses the printer through this interface. Also, if he wishes to use "Home dir access service", his agent acquires the HomeDir policy and invokes its client program. Then, the HomeDir client program tries to access the HomeDirLoginPass. Here, attributes of the HomeDirLoginPass are `Access=agent_attr:{owner="Home Directory Agent"}`, `Allowable_Communication="owner_only"`. Therefore, the HomeDir client program accesses the HomeDirLoginPass and tries to authenticate. If its authentication succeeds, userX can access his home directory; if it fails, he cannot. Of course, only programs generated by the Home Directory agent can access the HomeDirLoginPass .

4.4 The Evaluation of Our System

Service-Use Mechanism. We have defined pairs of service programs which are executed by a service provider agent and client programs which are executed by a user agent. A service provider agent provides service programs to user agents. Therefore, by getting client programs from service provider agents, user agents make use of various services. In the application example, if a user agent does not know a method for the service use in advance, he can use services by getting appropriate client programs from their service provider agents.

We also defined the tax policy, which is the obligation imposed on members of the community. Agents in the community must provide services specified in

tax policies. In the application example, the portal agent confirms whether an agent is a member of the laboratory or not by the Member_Check program, and the agents supplies the CPU resources for MPU simulation by the MPU_Simu program.

Protection of Private Resources. We defined two resource management spaces, the Public Zone and the Private Zone. The Public Zone is a space for flexible public service use. The Private Zone is a space for protecting private resources. All the access to resources in the Private Zone is examined by the Security Barrier according to each private policy. In the application example, we showed that only programs permitted by the private policy can access the LabID and the HomeDirLoginPass.

Decentralized Service Management. We introduce the community, which manages agents independently. In the application example, we defined the LabA and LabB community, each of which manages agents independently.

5 Related Work

UDDI (Universal Description, Discovery and Integration) , WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol) are three key specifications for implementing dynamic web service integration. UDDI offers users a unified and systematic way to find service providers through a centralized registry of services. WSDL is designed to describe the capabilities of any Web Service. SOAP is an XML-based protocol for messaging and Remote Procedure Calls (RPC). Because WSDL description is an interface for RPC or messaging, users must program to use its service. In our architecture, by getting a method for each service, user can use services without programming.

Many researchers are trying to develop security systems for Digital Rights Management and Trusted Computing[2, 4]. However, most of them are principally based on the client-server model or domain-specific distributed environments. Therefore, it is difficult to cover widely distributed environments in which there are a lot of services.

6 Summary

In this paper, we introduced a new architecture which has two resource management spaces: one is the Public Zone for flexible public service uses based on the acquisition of client programs; the other is the Private Zone where private resources are protected under the supervision of the Security Barrier. Therefore, a user agent can use various services while protecting private resources. Future work will clarify the details of the public/private policy attributes and the details of the client/service programs through practical application of the architecture.

References

1. distributed.net. http://distributed.net/.
2. J. S. Erickson. Fair Use, DRM, and Trusted Computing. *Communication of ACM*, Vol. 46(4), pp. 34–39, 2003.
3. T. Iwao, Y. Wada, M. Okada, and M. Amamiya. A Framework for the Exchange and Installation of Protocols in a Multi-Agent System. *CIA2001*, pp. 211–222, September 2001.
4. L. Kagal, T. Finin, and A. Joshi. Trust-Based Security for Pervasive Computing Environments. *IEEE Computer*, Vol. 34(12), pp. 154–157, 2001.
5. SETI@home. http://setiathome.ssl.berkeley.edu/.
6. Guoqiang Zhong, et al. The Design and Implementation of KODAMA System. *IEICE Transactions INF.& SYST.*, Vol. E85-D, No. 4, pp. 637–646, April 2002.

Online Performance Monitoring and Analysis of Grid Scientific Workflows*

Hong-Linh Truong[1] and Thomas Fahringer[2]

[1] Institute for Software Science, University of Vienna
truong@par.univie.ac.at
[2] Institute for Computer Science, University of Innsbruck
Thomas.Fahringer@uibk.ac.at

Abstract. While existing work concentrates on developing QoS models of business workflows and Web services, few tools have been developed to support the monitoring and performance analysis of scientific workflows in Grids. This paper describes a Grid service for performance monitoring and analysis of Grid scientific workflows. The service utilizes workflow graphs and various types of performance data including monitoring data of resources, execution status of activities, and performance measurement obtained from the dynamic instrumentation, to provide a rich set of monitoring and performance analysis features. We store workflows and their relevant information, devise techniques to compare constructs of different workflows, and support multi-workflow analysis.

1 Introduction

Recently many interests have been shown in exploiting the potential of the Grid for scientific workflows. Scientific workflows [12] are normally more flexible and diverse than production and administrative business workflows. As the Grid is diverse, dynamic and inter-organizational, even with a particular scientific experiment, there is a need of having a set of different workflows because (i) one workflow mostly is suitable for a particular configuration of underlying Grid systems, and (ii) available resources allocated for a scientific experiment and their configuration are changed in each run on the Grid. This requirement is a challenge for the performance monitoring and analysis of workflows (WFs) because very often the client of performance tools wants to compare the performance of different WF constructs with respect to the resources allocated in order to determine which WF construct should be mapped onto which topology of the underlying Grid. Therefore, multi-workflow analysis, the analysis and comparison of the performance of different WF constructs, ranging from the whole WF to a specific construct (e.g. a fork-join construct), is an important feature. Moreover, the performance monitoring and analysis of Grid scientific workflows must be conducted online. Even though numerous tools have been developed for constructing and executing scientific workflows on the Grid, e.g. [9, 14, 4], there is a lack of tools that support scientists to

* The work described in this paper is supported by the European Union through the IST-2002-511385 project K-WfGrid.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1154–1164, 2005.

monitor and analyze the performance of their workflows in the Grid. Most existing work concentrates on develop QoS models of business workflows and Web services [8, 3, 1].

To understand the performance of WFs on the Grid, we need to collect and analyze a variety of types of data relevant to the execution of the WFs from many sources. In previous work, we have developed a middleware which supports services to access and utilize diverse types of monitoring and performance data in a unified system named SCALEA-G [16]. This paper presents a Grid performance analysis service for scientific WFs. The analysis service, utilizing the unified monitoring middleware, collects monitoring data from the WF control and invocation services, and performance measurements obtained through the dynamic instrumentation of WF activities, and uses WF graphs to monitor and analyze the performance of WFs during the runtime. Relevant data of WFs including WF graphs and performance metrics are stored, and we develop techniques for comparing the performance of different constructs of WFs.

The rest of this paper is organized as follows: Section 2 outlines the Grid performance analysis service. Performance analysis for WFs is presented in Section 3. We illustrate experiments in Section 4. Section 5 discusses the related work. Finally we summarize the paper and outline the future work in Section 6.

2 Grid Performance Analysis Service

Figure 1 presents the architecture of the Grid monitoring and performance analysis service for WFs. The WF is submitted to the *Workflow Invocation and Control* (WIC) which locates resources and executes the WF. Events containing execution status of activities, such as *queuing, processing*, and information about resources on which the activities are executed will be sent to the monitoring tool. The *Event Processing* processes these events and the *Analysis Control* decides which activities should be instrumented, monitored and analyzed. Based on information about the selected activity instance and its consumed resources, the Analysis Control requests the *Instrumentation and Monitoring Control* to perform the instrumentation and monitoring. Monitoring and measurement

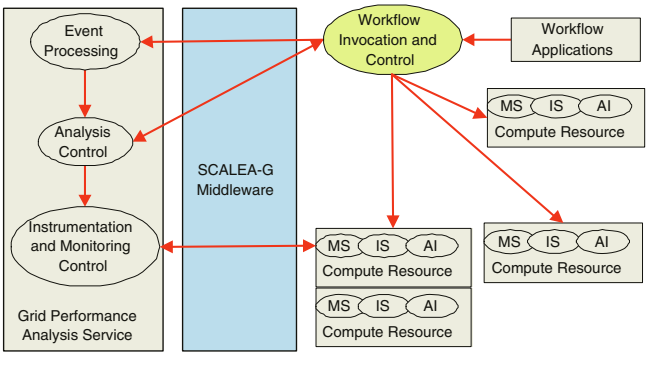


Fig. 1. Model of monitoring and performance analysis of workflow-based application

data obtained are then analyzed. Based on the result of the analysis, the Analysis Control can decide the next step. The performance monitoring and analysis service uses SCALEA-G as its supportive monitoring middleware. The monitoring service (MS) and Instrumentation Service (IS) are provided by SCALEA-G [16].

3 Performance Monitoring and Analysis of Grid Workflows

3.1 Supporting Workflow Computing Paradigm

Currently we focus on the WF modeled as a DAG (Direct Acyclic Graph) because DAG is widely used in scientific WFs. A WF is modeled as a DAG of which a node represents an activity (task) and an edge between two nodes represents the execution dependency between the two activities. An invoked application of an activity instance may be executed on a single or multiple resources.

We focus on analyzing (i) *fork-join* model and (ii) *multi-workflow* of an application. Figure 2(b) presents the fork-join model of WF activities in which an activity is followed by a parallel invocation of n activities. There are several interesting metrics that can be obtained from this model such as load imbalance, slowdown factor, and synchronization delay. These metrics help to uncover the impact of slower activities on the overall performance of the whole structure. We also concentrate on fork-join structures that contain *structured block* of activities. A structured block is a single-entry-single-exit block of activities. For example, Figure 2(c) presents structured blocks of activities.

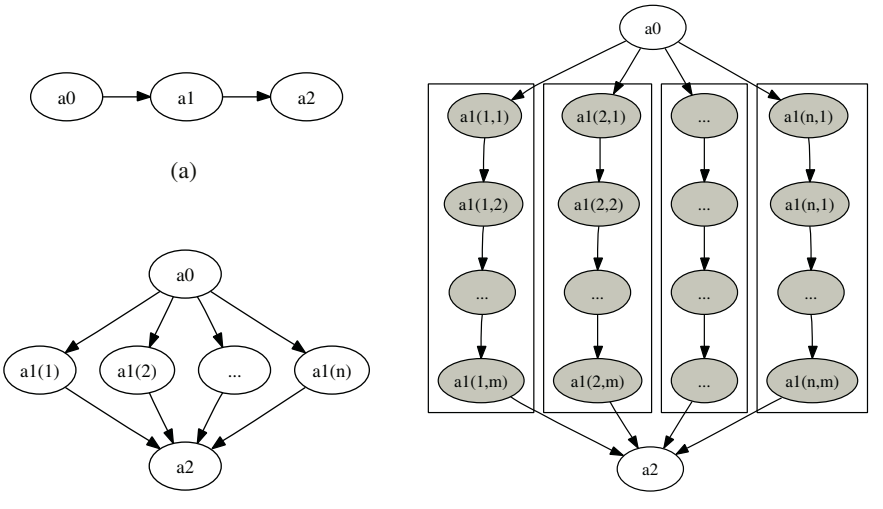


Fig. 2. Multiple workflows of an workflow-based application: (a) Sequence workflow, (b) Fork-join workflow, and (c) Fork-join of structured blocks of activities

A workflow-based application (WFA) can have different versions, each represented by a WF. For example, Figure 2 presents an application with 3 different WFs, each may be selected for execution on specific underlying resources. When developing a WFA, we normally start with a graph describing the WF. The WFA is gradually developed in a step-wise refinement that creates a new WF. In a refinement step, a subgraph may be replaced by another subgraph, resulting in a set of different constructs of the WF. For example, the activity $a1$ in Figure 2(a) is replaced by set of activities $\{a1(1), a1(2), \cdots, a1(n)\}$ in Figure 2(b).

We focus on the case in which a subgraph of a DAG is replaced by a another subgraph in the refined DAG. This pattern occurs frequently when developing WFs. Let G and H be DAG of WF WF_g and WF_h, respectively, of a WFA. G and H represent different versions of the WFA. H is said to be a *refinement* of G if H can be derived by replacing a subgraph SG of G by a subgraph SH of H. The replacement can be controlled by the following constraints:

- Every edge $(a, b) \in G$, $a \notin SG$, $b \in SG$ is replaced by an edge $(a, c) \in H$, $\forall c \in SH$ satisfies no $d \in H$ such that $(d, c) \in SH$.
- Every edge $(b, a) \in G$, $a \notin SG$, $b \in SG$ is replaced by an edge $(c, a) \in H$, $\forall c \in SH$ satisfies no $d \in H$ such that $(c, d) \in SH$.

SH is said to be a *replaced refinement graph* of SG. Note that SG and SH may not be a DAG nor a *connected graph*. For example, consider the cases of Figure 2(a) and Figure 2(b). Subgraph $SG = \{a1\}$ is replaced by subgraph $SH = \{a1(1), a1(2), \cdots, a1(n)\}$; both are not DAG, the first is a trivial graph and the latter is not connected graph. Generally, we assume that a subgraph SG has n components. Each component is either a DAG or a trivial graph. Comparing the performance of different constructs of a WFA can help to select and map WF constructs to the selected Grid resources in an optimal way.

Graph refinement is a well-established field and it is not our focus. We do not concentrate on the determination of refinement graphs in WFs, rather, the WF developers and/or WF construction tools are assumed to do this task. In this paper, (a_i, a_j) indicates the dependency between activity a_i and a_j, and *pred(a_i)* and *succ(a_i)* denote sets of the immediate predecessors and successors, respectively, of a_i.

3.2 Activities Execution Model

We use discrete process model [13] to represent the execution of an activity a. Let $P(a)$ be the discrete process modeling the execution of activity a. A $P(a)$ is a directed, acyclic, bipartite graph (S, E, A), in which S is a set of nodes representing *activity states*, E is a set of nodes representing *activity events*, and A is a set of edges representing ordered pairs of activity state and event. Simply put, an agent (e.g. WIC, activity instance) causes an event (e.g. submitted) that changes the activity state (e.g. from queuing to processing), which in turn influences the occurrence and outcome of the future events (e.g. active, failed). Figure 3 presents an example of a discrete process modeling the execution of an activity.

Each state s is determined by two events: leading event e_i, and ending event e_j such that $e_i, e_j \in E$, $s \in S$, and $(e_i, s), (s, e_j) \in A$ of $P(a)$. To denote an event *name* of $P(a)$

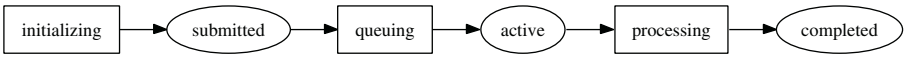


Fig. 3. Discrete process model for the execution of an activity. □ represents an activity state, ◯ represents an activity event

we use $e_{name}(a)$. We use $t(e)$ to refer to the timestamp of an event e and t_{now} to denote the timestamp at which the analysis is conducted. Because the monitoring and analysis is conducted at the runtime, it is possible that an activity a is on a state s but there is no such $(s, e) \in A$ of $P(a)$. When analyzing such state s, we use t_{now} as a timestamp of the current time of state s. We use $\rightarrow$ to denote the *happened before* relation between events.

3.3 Intra-activity and Inter-activity Performance Metrics

Performance data relevant to a Grid WF are collected and analyzed at two levels: *activity* and *workflow* level.

Activity Level. Firstly, we dynamically instrument code regions of the invoked application of the activity. We capture performance metrics of the activity, for example its execution status, performance measurements of instrumented code regions (e.g. wall-clock time, hardware metrics), etc. Performance metrics of code regions are incrementally provided to the user during the execution of the WF. Based on these metrics, various analysis techniques can be employed, e.g. load imbalance, metric ratio. We extend our overhead analysis for parallel programs [15] to WFAs. For each activity, we analyze *activity overhead*. Activity overhead contains various types of overheads, e.g. communication, synchronization, that occur in an activity instance.

Secondly, we focus on analyzing response-time of activities. *Activity response time* corresponds to the time an activity takes to be finished. The response time consists of waiting time and processing time. Waiting time can be queuing time, suspending time. For each activity a, its discrete process of execution model, $P(a)$, is used as the input for analyzing activity response time. Moreover, we analyze synchronization delay between activities. Let consider a dependency between two activities (a_i, a_j) such as $a_i \in pred(a_j)$. $\forall a_i \in pred(a_j)$, when $e_{completed}(a_i) \rightarrow e_{submitted}(a_j)$, the synchronization delay from a_i to a_j, $T_{sd}(a_i, a_j)$, is defined as

$$T_{sd}(a_i, a_j) = t(e_{submitted}(a_j)) - t(e_{completed}(a_i)) \quad (1)$$

If at the time of the analysis, $e_{submitted}(a_j)$ has not occurred, $T_{sd}(a_i, a_j)$ is computed as $T_{sd}(a_i, a_j) = t_{now} - t(e_{completed}(a_i))$. Each activity associates with a set of the synchronization delays. From that set, we compute maximum, average and minimum synchronization delay at a_j. Note that synchronization delay can be analyzed for any activity which is dependent on other activities. This metric is particularly useful for analyzing synchronization points in a WF.

Workflow Level. We monitor and analyze performance metrics that characterize the interaction and performance impact among activities. Interactions between two activities can be file exchanges, remote method invocations or service calls. In the analysis phase, we compute load imbalance, computation to communication ratio, activity usage, and success rate of activity invocation, average response time, waiting time, synchronization delay, etc. We combine WF graph, execution status information and performance data to analyze load imbalance for fork-join model. Let a_0 be the activity at the fork point. $\forall a_i, i = 1 : n, a_i \in succ(a_0)$, load imbalance $T_{li}(a_i, s)$ in state s is defined by

$$T_{li}(a_i, s) = T(a_i, s) - \frac{\sum_{i=1}^{n} T(a_i, s)}{n} \tag{2}$$

We also apply load imbalance analysis to a set of selected activities. In a WF, there could be several activities whose functions are the same, e.g. `mProject` activities in Figure 4, but are not in fork-join model.

3.4 Multi-workflow Analysis

We compute *slowdown factor* for fork-join model. Slowdown factor sf is defined by

$$sf = \frac{max_{i=1}^{n}(T_n(a_i))}{T_1(a_i)} \tag{3}$$

where $T_n(a_i)$ is the processing time of activity a_i in fork-join version with n activities and $T_1(a_i)$ is the processing time of activity a_i in the version with single activity. We also extend the slowdown factor analysis to fork-join structures that contain structured block of activities. In this case, $T_n(a_i)$ will be the processing time of a structured block of activities in a version with n blocks.

For different replaced refinement graphs of WFs of the same WFA, we compute *speedup* factor between them. Let SG be a subgraph of WF WF_g of a WFA; SG has n_g components. Let $P_i =< a_{i1}, a_{i2}, \cdots, a_{in} >$ be a critical path from starting node to the ending node of the component i, C_i, of SG. The processing time of SG, $T_{cp}(SG)$, is defined by

$$T_{cp}(SG) = max_{i=1}^{n_g}(T_{cp}(C_i)), T_{cp}(C_i) = \sum_{k=1}^{n} T(a_{ik}) \tag{4}$$

where $T(a_{ik})$ is the processing time of activity a_{ik}. Now, let SH be the replaced refinement graph of SG, SG and SH are subgraphs of WF WF_g and WF_h, respectively, of a WFA. Speedup factor sp of SG over SH is defined by

$$sp = \frac{T_{cp}(SG)}{T_{cp}(SH)} \tag{5}$$

The same technique is used when computing the speedup factor between WF_g and WF_h.

In order to support multi-workflow analysis of WFs, we have to collect and store different DAGs of the WF, performance data and machine information into an experiment

repository powered by PostgreSQL. Each graph is stored with its associated performance metrics; graph can be DAG of the WF or a subgraph. We use a table to represent relationship between subgraphs. Currently, for each experiment, the user can select subgraphs, specifying refinement relation between two subgraphs of two WFs. The performance tool uses data in the experiment repository to conduct multi-experiment analysis.

4 Experiments

We have implemented a prototype of the Grid performance analysis service with WIC is based on JavaCog [10]. JGraph [6] and JFreeChart [5] are used to visualize WF DAGs and performance results, respectively. In this section, we illustrate experiments of different WFs of the Montage application in the Austrian Grid [2].

Montage [11] is a software for generating astronomical image mosaics with background modeling and rectification capabilities. Based on the Montage tutorial, we develop a set of WFs, each generates a mosaic from 10 images without applying any background matching. Figure 4 presents experimental WFs of the Montage application. In Figure 4(a), the activity `tRawImage` and `tUncorrectedMosaic` are used to transfer raw images from user site to computing site and resulting mosaics from computing site to user site, respectively. `mProject` reprojects input images to a common spatial scale. `mAdd` coadds the reprojected images. `mImgtbl1` is used to build image table which is accessed by `mProject`, `mAdd`. In WFs executed on multiple resources, we have several subgraphs $tRawImage \rightarrow mImgtbl1 \rightarrow mProject1 \rightarrow tProjectedImage$, each subgraph is executed on a resource. The new `tProjectedImage` activity is used to transfer projected images produced by `mProject` to the site on which `mAdd` is executed. When executed on n resources, the subgraph $mImgtbl2 \rightarrow mAdd \rightarrow tUncorrectedMosaic$ is allocated on one of that n resources.

We conduct experiments on sites named GUP (University of Linz), UIBK (University of Innsbruck), AURORA6 (University of Vienna) and VCPC (University of Vienna) of the Austrian Grid. Due to the space limit, we just present a few experiments of online performance analysis of Montage WFs.

Figure 5 presents the performance analysis GUI when analyzing a Montage WF executed on two resources in UIBK. Performance analysis component retrieves profiling data through the dynamic instrumentation of invoked applications. The left-pane shows the DAG of the WF. The middle-pane shows the dynamic code region call graph (DRG) of invoked applications of activities. We can examine the profiling data of instrumented code region on the fly. The user can examine the whole DRG of the application, or DRG of an activity instance. By clicking on a code region, detailed performance metrics will be displayed in the right-pane. We can examine historical profiling data of a code region, for example window *Historical Data* shows the execution time of code region `computeOverlap` executed on `hafner.dps.uibk.ac.at`. The user also can monitor resources on which activities are executed. For example, the window *Forecast CPU Usage* shows the forecasted CPU usage of `hafner.dps.uibk.ac.at`.

Figure 6(a) presents the response time and synchronization delay analysis for activity *mImgtbl2* when the Montage WF, presented in Figure 4(c), is executed on 5 machines, 3 in AURORA6 and 2 in GUP. The synchronization delay from *tProjectedImage3, 4,*

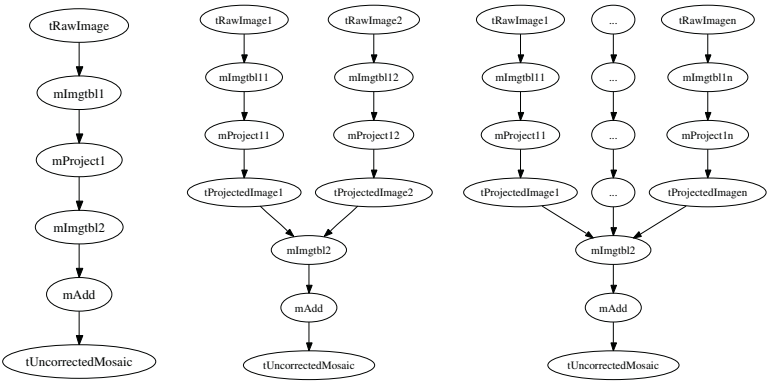


Fig. 4. Experimental workflows of the Montage application: (a) workflow executed on single resource, (b) workflow executed on two resources, and (c) workflow executed on n resources

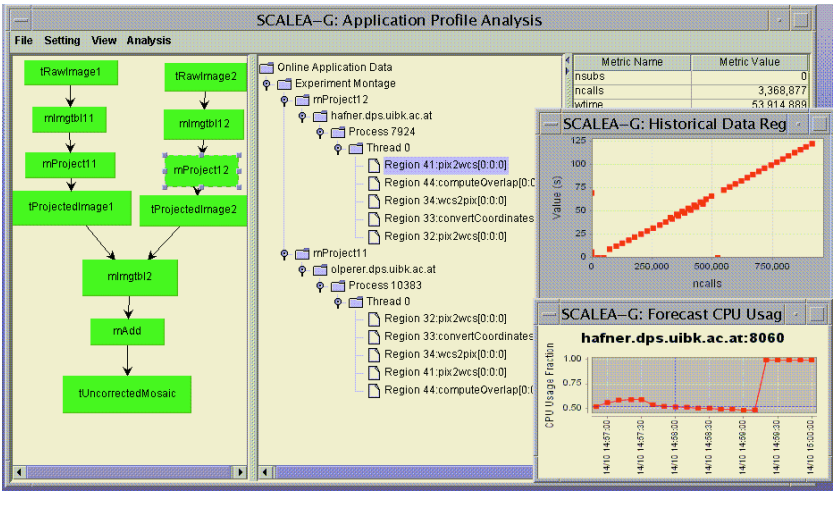
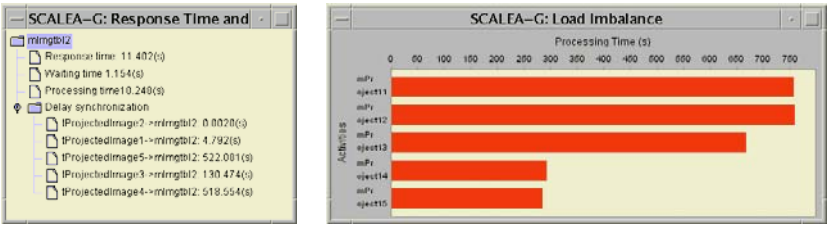


Fig. 5. Online profiling analysis for WF activities

5 to *tImgtbl2* are very high. This causes by the high load imbalance between *mProject* instances, as shown in Figure 6(b). The two machines in GUP can process significantly faster than the rest machines in AURORA6.

Over the course of the WF development process, subgraph named `mProjectedImage` which includes $tRawImage \rightarrow mImgtbl1 \rightarrow mProject1$ in single resource version is replaced by subgraphs of $tRawImage \rightarrow mImgtbl1 \rightarrow$



(a) (b)

Fig. 6. Analysis of Montage executed on 5 machines: (a) response time and synchronization delay of `mImgtbl`, (b) load imbalance of `mProject`

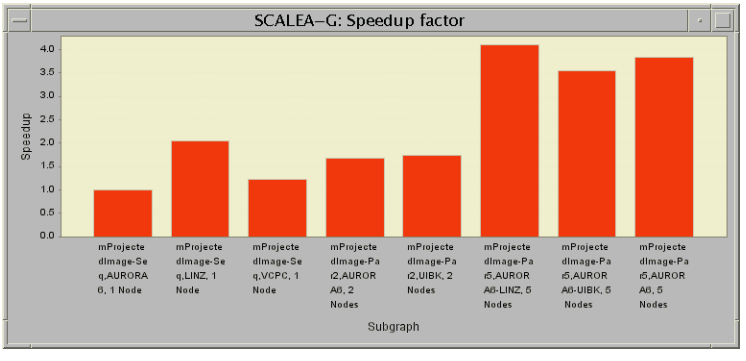


Fig. 7. Speedup factor for refinement graph `ProjectedImage` of Montage WFs

$mProject1 \rightarrow tProjectedImage$ in a multi-resource version. These subgraphs basically provide projected images to the `mAdd` activity, therefore, we consider they are replaced refinement graphs. We collect and store performance of these subgraphs in different experiments. Figure 7 shows the speedup factor for the subgraph `mProjectedImage` of Montage WFs executed on several experiments. The execution of `mProjectedImage` of the WF executed on single resource in LINZ is faster than that of its refinement graph executed on two resources (in AURORA6 or UIBK). However, the execution of `mProjectedImage` of WF executed on 5 resources, 3 of AURORA6 and 2 of LINZ, is just very slightly faster than that executed on 5 resources of AURORA6. The reason is that the slower activities executed on AURORA6 resources have a significant impact on the overall execution of the whole `mProjectedImage` as presented on Figure 6(b).

5 Related Work

Monitoring of WFs is an indispensable part of any WfMS. Therefore it has been discussed for many years. Many techniques have been introduced to study quality of service and

performance model of WFs, e.g., [8, 3], and to support monitoring and analysis of the execution of the WF on distributed systems, e.g. in [1]. We share them many common ideas and concepts with respect to performance metrics and monitoring techniques of the WF model. However, existing works concentrate on business WFs and Web services processes while our work targets to scientific WF executed on Grids. We support dynamic instrumentation of activity instances and online monitoring and performance profiling analysis of WFs, and integrate resources monitoring with WF monitoring.

Most effort on supporting the scientist to develop Grid workflow-based applications concentrates on WF language, WF construction and execution systems, but not focuses on monitoring and performance analysis of the Grid WFs. P-GRADE [7] is one of a few tools that supports tracing of WF applications. Instrumentation probes are automatically generated from the graphical representation of the application. It however limits to MPI and PVM applications. Our Grid WF monitoring and performance analysis service supports monitoring execution of activities and online profiling analysis. Also the dynamic instrumentation does not limit to MPI or PVM applications.

6 Conclusion and Future Work

This paper introduces a Grid performance analysis service that can be used to monitor and analyze the performance of scientific WFs in the Grid. The Grid performance analysis service which combines dynamic instrumentation, activity execution monitoring, and performance analysis of WFs in a single system presents a dynamic and flexible way to conduct the performance monitoring and analysis of scientific WFs. We believe techniques for comparing performance of subgraphs of WFs and for supporting multiple-workflow analysis are very useful for optimizing WF structures and mapping WF constructs onto selected underlying Grid resources.

In the current prototype, we manually instrument WIC in order to get execution status of activities. We can extend WF specification language with directives specifying monitoring conditions. These directives will be translated into code used to publish the status to the monitoring middleware. WIC can also offer an interface for the monitoring service to access that status. Meanwhile, the process of analysis, monitoring and instrumentation is controlled by the end-user. The future work is to automate that process.

References

1. Andrea F. Abate, Antonio Esposito, Nicola Grieco, and Giancarlo Nota. Workflow performance evaluation through wpql. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 489–495. ACM Press, 2002.
2. AustrianGrid. http://www.austriangrid.at/.
3. Jorge Cardoso, Amit P. Sheth, and John Miller. Workflow quality of service. In *Proceedings of the IFIP TC5/WG5.12 International Conference on Enterprise Integration and Modeling Technique*, pages 303–311. Kluwer, B.V., 2003.
4. Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, and Scott Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1:25–39, 2003.

5. JFreeChart. http://www.jfree.org/jfreechart/.
6. JGraph. http://www.jgraph.com/.
7. P. Kacsuk, G. Dozsa, J. Kovacs, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombas. P-GRADE: a Grid Programming Environment. *Journal of Grid Computing*, 1(2):171–197, 2003.
8. Kwang-Hoon Kim and Clarence A. Ellis. Performance analytic models and analyses for workflow architectures. *Information Systems Frontiers*, 3(3):339–355, 2001.
9. Sriram Krishnan, Patrick Wagstrom, and Gregor von Laszewski. GSFL : A Workflow Framework for Grid Services. Technical Report, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, U.S.A., July 2002.
10. G. Laszewski, I. Foster, J. Gawor, and P. Lane. A java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(643-662), 2001.
11. Montage. http://montage.ipac.caltech.edu.
12. Munindar P. Singh and Mladen A. Vouk. Scientific workflows. In *Position paper in Reference Papers of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-art and Future Directions*, May 1996.
13. John F. Sowa. *Knowledge Representation: logical, philosophical, and compuational foundations*. Brooks/Cole, Pacific Grove, CA, 2000.
14. The Condor Team. Dagman (directed acyclic graph manager). http://www.cs.wisc.edu/condor/dagman/.
15. Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Parallel Programs. *Concurrency and Computation: Practice and Experience*, 15(11-12):1001–1025, 2003.
16. Hong-Linh Truong and Thomas Fahringer. SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid. *Scientific Programming*, 2004. IOS Press. To appear.

WebGrid: A New Paradigm for Web System*

Liutong Xu[1], Bai Wang[1], and Bo Ai[2]

[1] School of Computer Science and Technology,
Beijing University of Posts and Telecommunications, Beijing 100876, China
{xliutong, wangbai}@bupt.edu.cn
[2] Information Systems Division of China Unicom, Beijing 100032, China
aibo@chinaunicom.com.cn

Abstract. World Wide Web is one of the most important applications on the Internet. Net surfers can browse the Web pages by tracing the links embedded in the hypertexts. However, one will be stuck when there are no hyperlinks or the hyperlinks are broken. This paper proposes a new paradigm for Web system called WebGrid, which consists of a dynamic collection of Web servers that work cooperatively. WebGrid synchronizes the Web contents that are usually stored in the underlying database systems in order to provide a uniform view of all resources. Net surfers can retrieve all resources in the WebGrid through one local Web server. The paper addresses WebGrid architecture and some related concepts. Key issues such as the WebGrid topology, the resource synchronization mechanism and strategy, as well as the Web site customization are also discussed in the paper.

1 Introduction

Computing paradigms have been evolved a lot since computer had been invented.

- 1^{st} *phase: Computer invented.* All computations are run on the local computer systems. There are no connections among different computer systems just as there were no roads in our world at the very beginning. It is impossible to utilize the computing power of some remote computer systems.
- 2^{nd} *phase: Internet invented.* Computers are connected by network. There are roads among the computer systems. Therefore, computations could be scheduled to run on some remote computer systems. However, you can do nothing if you do not know how to get to the destination.
- 3^{rd} *phase: Web invented.* Hyperlinks in the Web pages are just like the signposts that will guide you surfing on the Internet. In many situations, it is not necessary to know the destination addresses of the interested resources. If you are interested in something, you may click on the related links in the hypertext and you will get there.

* This work is supported by the National Natural Science Foundation of China (Grant No. 90104024), and China National Science Fund for Distinguished Young Scholars (Grant No. 60125101).

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1165–1174, 2005.

However, with the rapid development of the Internet application, the number of the Web sites and pages are growing up exponentially. What would happen if there were no signposts somewhere? Net surfers will be lost in the Web world with such an astronomical amount of Web pages.

A simplest approach to solve the problem is to set up appropriate signposts where necessary in the Internet. All these signposts together constitute a roadmap of the Web world. This brings a new computing paradigm.

– *4^{th} phase: Grid computing.* A complete roadmap never makes you lost. The Internet with the roadmap serves as a grid - a virtual supercomputer. Computations now run on grid.

This paper proposes a new paradigm called WebGrid. It unites a dynamic collection of Web servers to constitute a Web community and builds a roadmap for it. The resources in the WebGrid are well organized and the roadmap serves as a directory service. The goal of WebGrid is to realize the resource sharing among different Web sites and to give a unified resource view to the browsers. In the WebGrid, one can logon to one local Web server and retrieve all resources. For those non-local resources, the local server will retrieve the resources on behalf of the client from some remote Web servers and send them back to the client.

The remainder of this paper is organized as follows. Section 2 is about related work; section 3 proposes a WebGrid system and defines its architecture; section 4 addresses some key issues in WebGrid and detailed discussions; and section 5 presents some conclusions.

2 Related Work

World Wide Web is one of the most successful and popular ways to retrieve information on the Internet. Hypertext transfer protocol [1] brings up great convenience for the information retrieval on the Internet. People can trace the Web pages by clicking on the links in the hypertexts. On the other hand, we all had experienced trouble in finding information that we were interested in. The followings are some of the reasons:

– Sometimes you do not know the addresses where the resources are located, and you cannot find any hyperlinks that lead there.
– Not all information is linked properly. Sometimes the links are broken, especially those cross-site links.
– You would get no responses from the Web servers, because of the heavy workloads of the server or the network traffic jams.

Many efforts are done in order to leverage the performance of Web servers, to reduce the response time, to fasten the access speed.

Caching and mirror. Many organizations and ISPs have set up Web caching systems [7] or mirror sites over the different areas of the world in order for net surfers to visit their Web sites locally. Caching and mirror mechanisms can

speed up access to the Web servers, and balance the servers' workloads across the Internet. However, they cannot solve the problems of improper or broken cross-site links. Therefore, it cannot achieve resource sharing across Web sites from different organizations.

Search engine. If one has fewer knowledge about what one wants, search engine such as Google [5] seems to be a good choice. Google has a copy of all the world's Web pages and hopes net surfers to take it as a portal site to retrieve all the information on the Internet. Infomall project [6] in China has the similar goal with more than 700 million Web pages in its archive. However, almost everyone has experienced trouble with some search engines. Sometimes you get no results, sometimes you get flooded by tremendous amount of irrelevant or garbage information. In addition, the web pages in Google's database do not reflect the most up-to-date information. Moreover, the centralized search mechanism and database will lead to a bottleneck in practice.

The Semantic Web [2]. Its goal is to establish machine understandable Web resources. Researchers plan to accomplish this by creating ontology and logic mechanisms and replacing HTML with markup languages such as XML, RDF, OIL, and DAML. That means the semantic Web will rebuild radically rather than integrate conveniently the existing Web systems. Moreover, because the semantic Web, the semantic/knowledge grid [10] focus on semantics and reasoning, there is still a long journey to go before they will be widely deployed.

Grid technology [4] is to provide computing power as utility and realize the Internet scale resources sharing. Researchers hope to design some kind of generic grid infrastructures or platforms, and then to develop grid applications upon these platforms. On the other hand, our efforts focus on integrating some successful applications on the Internet to form application grids, such as FTPGrid [8]. In fact, the grid technology is a good approach to reconstruct the Web system. Nowadays, most of Web servers on the Internet have adopted dynamic Web technologies, which means most of information are stored in the underlying database systems. If we can construct a data grid [3] to store all information of some related Web sites, then all the Web sites can be reconstructed based on the data grid. In this paper, we are going to unite the Web servers to constitute a Web grid system - WebGrid. Obviously, retrieving information in a well organized WebGrid is always better than searching in a chaotic Web world.

3 The WebGrid System

With the rapid development of the Internet applications, more and more Web servers have adopted dynamic Web page technologies, which means Web information are stored in the underlying database systems. However, fewer resources are shared among these Web servers. The goal of WebGrid is to achieve information sharing and fully utilize the Web resources that are stored in different Web sites. A WebGrid consists of a dynamic collection of geographically distributed Web servers (also called grid node). The unified view of all Web resources is established by synchronizing the underlying database systems. Thus, one can

browse all information of these Web servers easily without following a series of hyperlinks jumping back and forth.

3.1 Two Typical Scenarios

What does WebGrid look like? Let us look at two scenarios. Here we take news Web site for example. Normally, all news agencies have their own Web sites. The journalists write news reports and publish on their news Web sites.

A Traditional Web Scenario. If you want to read some daily news reported by CNN, for example, you must logon to the CNN Web site. If you want to read news reported by BBC, then you must logon to the BBC Web site. As shown in Fig. 1. However, if you have never heard of some news agency, Fox for example, then there is no way for you to read Fox news. You cannot connect to it or even you do not know how and what to search. In addition, you may read many reprints of the same pieces of reports on the different sites in the traditional Web scenario.

A WebGrid Scenario. Suppose that the most important mass media such as CNN, BBC, Fox, CCTV, etc. have agreed on sharing their news reports. Now,

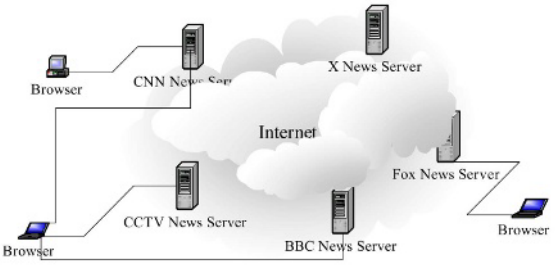


Fig. 1. A traditional Web scenario

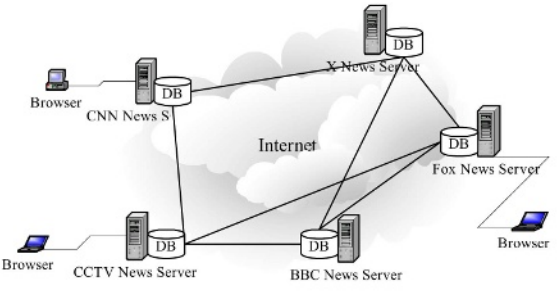


Fig. 2. A WebGrid scenario

they synchronize their news reports databases to form a virtual news database. All these Web sites together form a NewsGrid, as shown in Fig. 2, a virtual unified news Web server. Now, you only need to logon one of the Web sites in the NewsGrid, and can browse all reports.

3.2 WebGrid Architecture

The original Web adopted so-called two tiers Browser/WebServer architecture. Recently more and more Web sites are adopting dynamic Web technology and information is stored in the underlying database systems. This is three tiers Browser/WebServer/DatabaseServer architecture. If we unite a group of Web servers by integrating the underlying database systems, we get a virtual database and further a virtual Web server or a WebGrid. Fig. 3 describes the WebGrid architecture. The two lower tiers construct a grid tier. Therefore, it can also be viewed as two tiers Browser/Grid architecture. People can browse a WebGrid conveniently just as browsing a traditional Web site, because the underlying database systems are transparent to the WebGrid users.

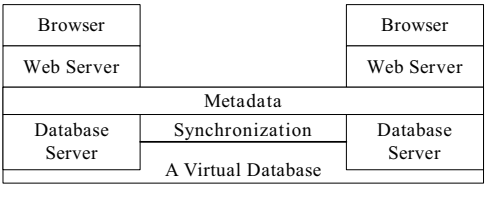


Fig. 3. WebGrid architecture

3.3 Metadata

The metadata takes an important role in the grid environment. It addresses important information about the grid resources, such as name, ID, owner, description, locations, etc. In the NewsGrid, for example, the resources are mainly the news reports, the metadata may include:

- ID, title, description, length
- Type: political, financial, entertainment, or sports, etc.
- Region: such as domestic or international
- Authority or not
- Locations: multiple copies of resources are stored in the WebGrid. Therefore, Web resource may have multiple locations either in local database or in remote databases, or in file systems
- Timestamp: used for updating
- etc.

In the WebGrid, unified and consistency metadata is necessary to give a uniform view of all the resources.

Authority Resource. Because WebGrid consists of many Web servers, the Web resources are owned by different organizations. The resources will be classified into two categories: authority and non-authority. The resources that have important or historical values are called authority resources and should be kept in the archives. The resources such as those in caches or for temporal use are non-authority and may be deleted upon the limitation of the local storages and the replacement policies.

Whether the resources are authority or not is site-relevant. Every Web site may have its own policies to identify some of its resources as authority or non-authority. For example, the news reports written by CNN journalists are authority to the CNN Web sites, but non-authority to the CCTV Web sites, and vice versa.

3.4 The Way WebGrid Works

WebGrid adopts so-called browser/grid architecture. Web browsers only need to connect to one local Web server, and then they can get access to all resources in the WebGrid.

Take NewsGrid for example. If someone in China wants to read news about the tsunami/earthquake occurred recently in Indian Ocean islands, he/she can logon to CCTV Web site. The server will return a Web page containing the related news reports that are generated from the news database. Some of them are reported by CCTV, others by CNN or CBS. If the report is reported by CCTV or it is reported by CNN but it has been requested previously and has been cached in the local database system, then the Web server will access the report from the local database and return to the browser. If it is a CBS report and there is no cached copy, then the Web server will retrieve on behalf of the browser the corresponding news report from some remote database systems, and then return the requested report to the browser. At the same time, the report will be cached in the local database system for subsequent requests.

4 Key Issues in the Implementation of WebGrid

A WebGrid consists of a dynamic collection of geographically distributed Web servers (grid nodes) that work cooperatively as a single one. For WebGrid to be a reality, some key issues must be solved.

The resource management is the key issue in the grid environment. In the WebGrid environment, the resource management is mainly about the synchronization of the Web resources that are stored in the underlying database systems and of the metadata that describe the Web resources. Because of the complexity of the WebGrid topology, the neighborhood-based policy is introduced to simplify the synchronization. The metadata will be synchronized completely in order to give a unified view of all the resources, while the Web resources only need to be partially synchronized. Other issues in WebGrid include access relay

and caching, join and leave of a Web server, customization of individual Web servers, etc.

4.1 WebGrid Topology and Neighborhood

WebGrid is directly built upon the existing Web sites that may cover wide area over the Internet. Therefore, its topology is quite complicated. The resource management in such an environment will become very difficult. In order to simplify the resource management, we introduce the concept of *neighborhood*. The resource management in the WebGrid will be based on the neighborhood.

Distance. The neighborhood is measured by some kind of distances. Distance between two grid nodes can be any one or combination of the followings (perhaps with some kind of weights):

- network hops, network bandwidth capacity, real data transfer rates (which varies dynamically with time), round-trip time, data transfer costs, etc.

For a specific Web site (node), other Web sites can be classified into three types, according to the distance between the two sites.

- *Near Neighbor*: a node with relatively short distance from the node
- *Distant Neighbor*: a node with quite long distance from the node
- *Non-Neighbor*: a node that is too far from the node

Neighborhood Topology. In the WebGrid, every grid node has its own neighborhood.

- Every grid node keeps the topology information about its neighborhood, including near neighbors and sometimes distance neighbors.
- Every grid node manages resource sharing and information synchronization by exchanging information with neighboring nodes.
- Because the synchronization is based on neighborhood, it can be implemented in a decentralized manner. Therefore, the WebGrid is scalable from a small organization to the Internet scale.

4.2 Resource Synchronization

WebGrid consists of many Web sites distributed geographically over the Internet. The resource management in the WebGrid is implemented by synchronizing the underlying databases. The WebGrid topology is very complicated. Therefore, in order to simplify and achieve the decentralized resource management in the WebGrid, every Web site synchronizes its resources only with its neighboring sites. The metadata describe the information about all resources stored on the geographically distributed Web servers, so it is necessary to have a complete metadata to present a unified resource view to all Web browsers. On the other hand, it is not necessary for all Web servers to keep a complete copy of all

resources in the WebGrid, or it will become a mirroring mechanism and obviously consume too many storage resources. In fact, each Web server only need to store the authority resources and those resources that are frequently accessed by the local browsers.

Therefore, different synchronization mechanisms are used in WebGrid for different purposes:

Topology Information Exchange. In WebGrid, all resource synchronization is based on neighborhood, so the neighborhood topology information are exchanged and updated periodically among the neighboring nodes.

Metadata Synchronization. Metadata must be synchronized completely to present a unified view of the WebGrid resources. However, the metadata synchronization does not mean to have identical copies of metadata at all Web sites. Some attributes about the resources may be different for different Web sites. The resources may be local or authority to some Web sites, but remote or non-authority to other Web sites.

Resource Synchronization. The Web resources in the WebGrid are all stored in the underlying databases. It is not necessary to synchronize completely all these databases. Only the resources that are requested frequently by the local browsers are needed to be synchronized from its neighboring sites to the local server. While those resources that are not so frequently requested will be retrieved on demand from some neighboring nodes.

Information Update. We all experienced the delayed update in information or the broken links. In the WebGrid, the owners of the resources take the responsibility to update information or links in hypertexts. When new information is inserted or some information is updated in the local database, it will trigger metadata resynchronization with its neighbor sites, and then spread over the whole WebGrid. Therefore, the metadata are kept up to date.

4.3 Access Relay and Caching

Because the partial synchronization mechanism is used in the WebGrid, sometimes the resources that a browser requested are not stored in the local database system. At that time, the local Web server will access the resources from some remote database systems according to metadata, and then send the resources back to the browser. This access relay mechanism combined with the partial synchronization mechanism forms the key foundation of the WebGrid system. In other words, any requests to the non-local resources will trigger a data replication process to implement the resource synchronization. A proper data replication strategy such as [9] is used in order to support fragmented replication from multiple source sites.

The resources retrieved from the remote database systems will be stored temporally on the local database system in order to speed up the response time for possible subsequent requests. Usually, the temporal copies of the resources will be marked up as non-authority, which means the resources could be deleted after certain period. The period depends on how frequently the resources have been accessed in the most recent period and the local site's storage limitation.

4.4 Join and Leave of Web Servers

In the dynamic environment of WebGrid, it is quite often that some servers will join in or leave from a grid. If a Web server joins into a WebGrid, the newcomer should first synchronize metadata from its neighbor Web servers. When a Web server leaves, all the resources including those authority resources stored on the server will become unavailable. The neighboring nodes will update their topology information and metadata periodically. In addition, the WebGrid could also keep several copies of such authority resources on other nodes upon certain policies.

4.5 Customization

Although the WebGrid has a unified view of all the resources, the Web sites in the WebGrid can still have different appearances. That means each Web site can be customized based on some site-specific information such as private information, language setting, layout and color designing, and preference list of information stored in the local database, etc.

For example, in the NewsGrid above, the news reports may be divided into two classes: domestic and international. Of course, it depends on the countries that the Web sites reside. The events happened in the US would be listed in the domestic news in CNN or CBS Web sites, but listed in the international news in BBC or CCTV Web sites. The reports about Chinese Spring Festival are domestic to CCTV Web site and international to CNN and BBC Web sites. CCTV will put the political news at the most significant place in the layout, while CNN might put the financial news at that place.

4.6 Advantages of WebGrid

If we unite a collection of Web servers to form a WebGrid, we may experience many benefits. The advantages include:

- *Convenient access:* The underlying technology of WebGrid is transparent to the Web browsers, so net surfers can browse the WebGrid as if they were browsing the traditional Web sites.
- *Unified view:* All resources are integrated to form a unified view of all Web sites. The neighborhood-based synchronization mechanism realizes decentralized resource management, and therefore WebGrid is scalable.
- *Load balance:* Every Web server in the WebGrid can be considered as a cache of other Web servers. Therefore, the access load will be distributed evenly over the whole WebGrid.

- *Fast access:* Local access will provide much higher access speed.
- *Reduce network traffic:* Many remote accesses are replaced by the local accesses, so the network traffic on the backbone will be reduced.
- *Customization:* Each Web server may be customized to have different layout, language, color, preferences, etc.

5 Conclusions

This paper presented a new paradigm for Web system - WebGrid, which consists of a dynamic collection of geographically distributed Web servers working cooperatively for all Web browsers. People can browse the WebGrid as if they are browsing the traditional Web system.

WebGrid makes it possible for the Web browsers to get a convenient and a high-speed access to all resources in the WebGrid through a local Web server. The neighborhood-based synchronization mechanism simplifies the resource management in WebGrid. The complete synchronized metadata and the partial synchronization mechanism for the Web contents give a global view of the WebGrid, and reduce the storage costs by storing Web resources only on their authority servers and some hotspot servers. WebGrid can radically reduce the traffic on the network, because the same resources will not be transferred back and forth frequently on the backbone of the Internet. In WebGrid, every Web server may be customized to fit its local browsers' preferences.

References

1. Berners-Lee, T., Fielding, R., and Frystyk, H.: Hypertext Transfer Protocol - HTTP/1.0. IETF, RFC 1945. http://www.ietf.org/rfc/rfc1945.txt (1996)
2. Berners-Lee, T., Hendler, J. and Lassila, O.: The Semantic Web. Scientific American. 284(5)(2001) 34-43
3. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S.: The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. J. Network and Computer Applications. 23 (2001)
4. Foster, I. and Kesselman, C. (eds.): The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, San Francisco (2003)
5. Google. http://www.google.com
6. Infomall Project. http://www.infomall.cn
7. Wessels, D. and Claffy, K.: Internet Cache Protocol (ICP), version 2. IETF, RFC 2186. http://www.ietf.org/rfc/rfc2186.txt (1997)
8. Xu, L. and Ai, B.: FTPGrid: A New Paradigm for Distributed FTP System. In: Li, M. et al (Eds.): Grid and Cooperative Computing. Lecture Notes on Computer Science, Vol. 3033. Springer-Verlag, Berlin Heidelberg (2004) 895-898
9. Xu, L., Wang, B. and Ai, B.: A Strategy for Data Replication in Data Grids. In: Proc of the 2004 Int. Conf. on High Performance Computing and Applications. Shanghai. To appear in LNCSE, Springer-Verlag (2005)
10. Zhuge, H.: China's E-Science Knowledge Grid Environment. IEEE Intelligent Systems. 19(1) (2004) 13-17

Dynamic Failure Management for Parallel Applications on Grids

Hyungsoo Jung, Dongin Shin, Hyeongseog Kim, Hyuck Han,
Inseon Lee, and Heon Y. Yeom

School of Computer Science and Engineering,
Institute of Computer Technology,
Seoul National University,
Seoul, 151-742, Korea
{jhs, dishin, hskim, hhan, inseon, yeom}@dcslab.snu.ac.kr

Abstract. The computational grid, as it is today, is vulnerable to node failures and the probability of a node failure rapidly grows as the size of the grid increases. There have been several attempts to provide fault tolerance using checkpointing and message logging in conjunction with the MPI library. However, the Grid itself should be active in dealing with the failures. We propose a dynamic reconfigurable architecture where the applications can regroup in the face of a failure. The proposed architecture removes the single point of failure from the computational grids and provides flexibility in terms of grid configuration.

1 Introduction

Grid computing is an emerging computing paradigm which utilizes a heterogeneous collection of computers in different domains connected by networks[1]. There are computational grid which is mainly for high performance computing and data grid where enormous amount of data are spread out throughout the grid. Though the grid has attracted much attention for its ability to utilize ubiquitous computational resources while providing a single system view, most of grid-related studies have been concentrated on the static resource management. The dynamic resource management, especially the fault tolerance has not been discussed enough in the grid research area. The grid, being an instance of distributed systema, has inherent failure factors; the system consists of a number of nodes, disks and network lines which can fail independently. Although the failure rate of a single component could be acceptable, the overall system reliability of the system consisting of various components can be suprisingly low if all the components should be in working order for the overall system to be functional. To increase the reliability of the system, it is essential to provide some form of fault tolerance. Also, the grid, being a collection of various nodes and clusters, should be able to cope with changing configuration due to failures.

In this paper, we propose a framework for dynamic failure management for the computational grid where parallel programs are executing on Globus. The

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1175–1182, 2005.

most important issue in providing fault tolerance is that there should not be a single point of failure. We have achieved this goal using different strategies at different levels accordingly. At the lowest level, where the application programs are concerened, we have employed checkpointing and message logging to store the applications' consistent global states so that the failed process can be restarted on another node without losing the entire computation. Checkpointing and rollback recovery is a well-known technique for fault tolerance, which is an operation to store the state of a process into stable storage so that the process can resume its previous states at any time [2]. At the next level, where the application processes are managed, state machine approach[3] is employed by active replication of cluster managers.

Although several stand-alone checkpoint toolkits are present [1], they are not adquate for parallel applications since they cannot restore the communication context: for examble, sockets or shared memory content would be wiped out. Since parallel processes communicate with each other by message passing, the communication channels should be restored as well as the process states. In this paper, we concentrate on the channel reconstruction for parallel processes restored from checkpoints.

2 Related Works

Several implementations of the fault tolerance have been proposed, which exploits the dynamic process management of PVM [4] or LAM-MPI [5, 6, 7]. Such parallel programming environments allow flexible change of the process group, that is to say, the processes can leave and join the process group dynamically. However, these implementations are based on the indirect communication model where the messages are transferred via intermediates like daemons. Each process is not required to know the physical addresses of the other processes. A process only needs to know its corresponding daemon to communicate and channel reconstruction can be easily done by reconnecting the local daemon. However, indirect communication imposes a large message delay which undermines the biggest advantage of MPI, the performance.

MPICH-G2 proposed by Argonne National Laboratory [8] is a grid-enabled MPICH that runs on Globus middleware [1]. To the best of our knowledge, MPICH is the most popular MPI implementation for its good performance and portability. Each MPI process is aware of the physical channel information and it can send messages to the target process directly. Once the process group set is fixed at the initialization, no additional process can join the group. Hence, a process failure results in total failure and the job has to be resubmitted from the beginning. Actually original MPI (ver 1.x) specifies only the static process management.

[1] For more information about the checkpointing toolkits available on Internet, we would like to recommend the web site, *http://www.checkpointing.org*

To cope with this, we proposed `MPI_Rejoin()` [9] for dynamic MPI process management. The rejoin operation enables the failed process to rejoin the existing group after recovery by updating the corresponding entry of the channel table with the new physical address. It is the foundation of MPICH-GF [10], which is our fault tolerant implementation of MPICH for the Grid system running Globus. The MPICH-GF provides fault tolerance by using checkpoint and roll-back recovery in conjunction with the message logging.

However, failure detection and recovery also should be handled in fault tolerant fashion. To detect the failure and coordinate the recovery, a hierarchical process managers are employed: the central manager which has total control over all other processes and the local managers to deal with each application processes.

The rest of this paper is organzied as follows. In Section 3 and Section 4, we describe the communication mechanism of MPICH-G2 and the MPICH-GF architecture respectively. We prsents the design and implementation of `MPI_Rejoin()` in Section 5. We also show the experimental results in Section 6. In the final section, we conclude and propose future works.

3 Communication Mechanism of MPICH-G2

Good portability of MPICH can be attributed to the abstraction of low-level operations, the Abstract Device Interface (ADI), as shown in Figure 1. An ADI's implementation is called a virtual device. Especially MPICH with a grid device *globus2* is called MPICH-G2 [8]. Just for reference, our MPICH-GF has been implemented as the unmodified upper MPICH layer and our own virtual device *ft-globus* based on *globus2*.

The communication of MPICH-G2 is based on non-blocking TCP sockets and active polling. In order to accept a request for channel construction, each MPI process opens a *listener port* and registers it on the file descriptor set with callback function to poll it. On the receipt of a request, the receiver opens another socket and accepts the connection. Then both processes enter the *await_instruction* state where they exchange the metadata of the connection points. During this channel construction, the process with the larger rank ID as-

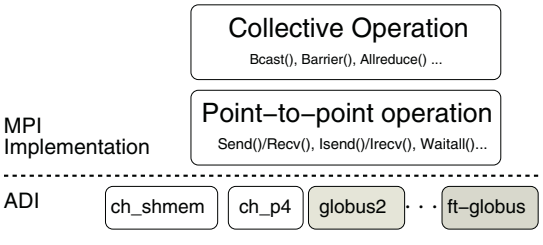


Fig. 1. Architecture of the MPICH

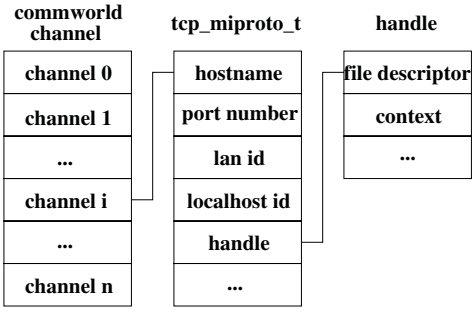


Fig. 2. Description of `MPI_COMM_WORLD`

sumes the role of a master whether it is a sender or a receiver. Every process gets to know the listener ports for all processes after initialization. The initialization in MPICH-G2 consists of two main procedures: the rank resolution and the exchange of listener information. Rank is the logical process ID in the group that is determined through the check-in procedure. Check-in procedure is performed in two phases: the *intra check-in* in local nodes and the *inter check-in* in global area. This is done with the help of hierarchical manager system, which is described in the next section. After the check-in procedure, the master process with rank 0 becomes aware of other processes' ranks and broadcasts it. Then, each process creates the listener socket and exchanges its address with the siblings processes to construct the `MPI_COMM_WORLD`.

Figure 2 presents `MPI_COMM_WORLD`, the data structure containing the listener and channel information for all processes. The ith entry contains the channel information of process with rank i as follows:

- *hostname* and *port* are the phyiscal address of the listener.
- *handle* constains the file descriptor information of the target channel. If the channel has not constructed yet, the value is null.
- *lan ID* and *localhost ID* are the network domain information.

4 MPICH-GF

MPICH-GF [10] is our own fault tolerant MPICH implementation on grids that is originated from MPICH-G2. It supports coordinated checkpointing, sender-based message logging and the optimistic receiver-based message logging [2] in order to guarantee consistent recovery. We have implemented those recovery algorithms based on the user-level checkpoint library and the process rejoining module described in Section 5. Our MPICH-GF implementation is achieved at the virtual device level only and it does not require any modification of application source code or the MPICH sublayer upon ADI. It also guarantees that both blocking and non-blocking in-transit messages during checkpointing are never

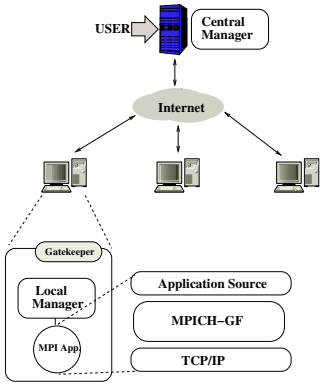


Fig. 3. Process management architecture of Globus

lost. MPICH-GF is aided by the hierarchical management system which is responsible for the failure detection and the process recovery. The central manager and the local managers in Figure 3 are based on Globus resource co-allocator and GRAM job managers [11] repectively.

5 Design of `MPI_Rejoin`

We have devised `MPI_Rejoin()` function in order to allow the recovered process be recognized as the previous process instance. `MPI_Rejoin()` was designed based on the `MPI_Init()` which is called at the beginning of the MPI application to initialize the MPI communication information. The major difference is that we want to retain survived processes' communication information intact while updating infortion regarding the restored process' communication channel. MPICH-G2 has many reference variables related to the system level information as shown in Table 1. These variables are used by select() system call for non-blocking TCP communication. The variables except the final entry constains the input arguments and return values of *select()* system call.

The read / write callback functions are stored in "globus_l_io_fd_table". Since these sytem level information are not valid any more in a new process instance, they should be re-initiated in order to prevent a recovered process from accessing the invalid file descriptors. Then, the new physical address of the restored process should be notified to the other processes. Figure 4 shows the rejoin process incurred by calling `MPI_Rejoin()`. After reopening the listener socket, the restored process sends its new listener information (Table 2) to the local manager. Then the local manager forwards this information to the central manager. The central manager collects all channel update information from the restored processes and finally broadcasts them. On the receipt of the update information, the survived processes update the corresponding entries of their `MPI_COMM_WORLD`.

Table 1. Reference variables

globus_l_io_read_fds
globus_l_io_write_fds
globus_l_io_except_fds
globus_l_io_active_read_fds
globus_l_io_active_write_fds
globus_l_io_active_except_fds
globus_l_io_highest_fd_num_set
globus_l_io_select_count
globus_l_io_fd_num_set
globus_l_io_fd_table

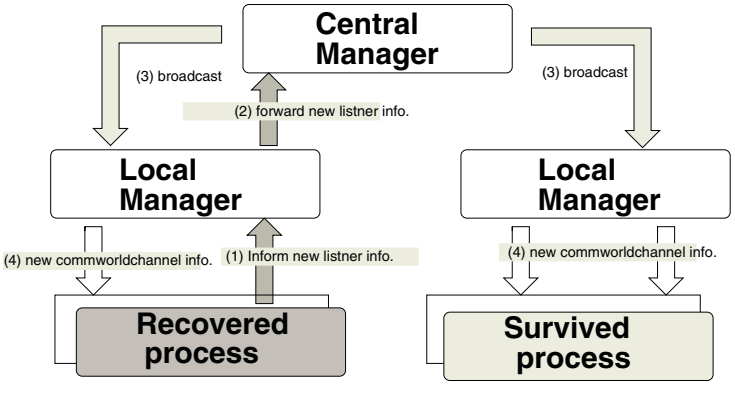


Fig. 4. Protocol between Manager and Processes

Table 2. Rejoin message format

global rank (4 Bytes)	hostname (256 Bytes)	port number (10 Bytes)	lan id (4 Bytes)	localhost id (40 Bytes)

The values of *handle* in those entries are set to null, which makes the processes to reconstruct the channel.

6 Experimental Results

We have tested our implementation using a testbed of eight computers: Four with Intel Pentium 800 MHz processor and four with Intel Pentium 2.0 GHz processor. All of them have 128 MB RAM with Linux-2.4.12 OS and Globus

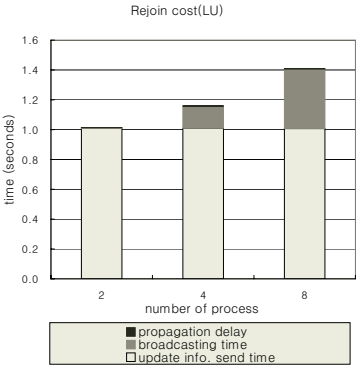


Fig. 5. Rejoin Overhead

toolkit 2.2 installed. The NAS Parallel Benchmark applications were used to verify our implementation. Since we are only interested on the performance of recovery process and there is little difference between different applications, we only presents the results from running LU application. All the other applications show similar behavior when recovering. In this experiment, the number of processes varies from two to eight.

The *"rejoin"* cost per single failure is presented in Figure 5. In order to evaluate the *"rejoin"* cost, we have measured the "Update info. send time", "Broadcasting time" and "Propagation delay". "Update info. send time" is the time for the recovering process to initialize the communication channel and send the new channel information to the local manager. It was measured after the recovering process was re-launched and restored by reading the checkpoint file. It contains the reference variable re-initialization overhead, new listener port creation overhead and the communication overhead to the local manager. Since this cost is not influenced by the number of processes which participate in the current working group, it should not vary according to the number of processes and the result matches with the expectation. The "Broadcasting time" was measured at the central manager as the time between receiving the recovering process' channel update informatioin and receiving the acknowledgement from all proceses in current working group. Figure 5 shows that "Broadcasting time" increases as the number of processes increases. After all processes in current working group receive the update signal from the central manager, they modify *"MPID_COMM_WORLD"* and send the acknowledgement for notificatoin of channel update complete at each process to central manager. "Propagation delay" contains the message delay from central manager to local manager, from local manager to central manager and from local manager to MPI process. This overhead is pretty small and almost constant. As shown in Figure 5, only "Broadcasting time" is influenced by the number of processes.

7 Conclusion

We have presented the design and implementation of dynamic process management using process rejoining for MPICH on grids. The proposed operation is the basis of providing fault tolerance and it can also be used for task migration for message passing processes. While most of previous works adopt the indirect communication for the convenience of the channel reconstruction, our MPICH-GF supports the direct communication model for good performance. In order to respect the communication characteristic, our implementation is accomplished at the virtual device layer. However, it was still accomplished at user level without modifying the Linux kernel. We have evaluated the cost of `MPI_Rejoin()` with varying process group size and show that the cost increases as the number of processes increases. We expect that on-demand channel updating would be more efficient since each parallel process tends to communicate with specific adjacent processes. On-demand in this context means that the process specifically ask the channel information when needed instead of central manager broadcasting the information to all the processes.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International J. Supercomputer Applications **15** (2001)
2. Elnozahy, E.N., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys **34** (2002) 375–408
3. Schneider, F.B.: The state machine approach: a tutorial. (1986)
4. Menden, J., Stellner, G.: Proving properties of pvm applications - a case study with cocheck. In: PVM/MPI 1996. (1996) 134–141
5. Li, W.J., Tsay, J.J.: Checkpointing message-passing interface (MPI) parallel programs. In: Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems (PRFTS). (1997) 147–152
6. Fagg, G.E., Dongarra, J.: FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In: PVM/MPI 2000. (2000) 346–353
7. Louca, S., Neophytou, N., Lachanas, A., Evripidou, P.: Portable fault tolerance scheme for MPI. Parallel Processing Letters **10** (2000) 371–382
8. Foster, I., Karonis, N.T.: A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In: Proceedings of SC 98, ACM Press (1998)
9. Kim, S., Woo, N., Yeom, H.Y., Park, T., Park, H.: Design and implementation of dynamic process management for grid-enabled MPICH. In: Proceedings of the 10th European PVM/MPI Users' Group Conference. (2003)
10. Woo, N., Yeom, H.Y., Park, T., Park, H.: MPICH-GF, transparent checkpointing and rollback-recovery for grid-enabled MPI processes. In: Proceedings of the 2nd Workshop on Hardware/Software Support for High Performance Scientific and Engineering Computing. (2003)
11. Foster, I., Kesselman, C.: The globus project: A status report. In: Proceedings of the Heterogeneous Computing Workshop. (1998) 4–18

A Novel Intrusion Detection Method for Mobile Ad Hoc Networks

Ping Yi*, Yiping Zhong, and Shiyong Zhang

Department of Computing and Information Technology Fudan University,
Shanghai, 200433, China
Pyi_edu@yahoo.com.cn

Abstract. The mobile ad hoc networks are particularly vulnerable to intrusion, as its features of open medium, dynamic changing topology, cooperative routing algorithms. The traditional way of protecting networks with firewalls and encryption software is no longer sufficient and effective for those features, because no matter how secure the mobile ad hoc networks, its is still possible the nodes are compromised and become malicious. In this paper, we propose a novel intrusion detection approach for mobile ad hoc networks by using finite state machine. We construct the finite state machine (FSM) by the way of manually abstracting the correct behaviours of the node according to the routing protocol of Dynamic Source Routing (DSR). The monitor nodes cooperatively monitor every node's behaviour by the FSM. Our approach can detect real-time attacks without signatures of intrusion or trained data. Finally, we evaluate the intrusion detection method through simulation experiments.

1 Introduction

Mobile Ad Hoc Networks are the collection of wireless computer, communicating among themselves over possible multi-hop paths, without the help of any infrastructure, such as base stations or access points. Nodes in mobile ad hoc network collaboratively contribute to routing functionality by forwarding packets for each other to allow nodes to communicate beyond direct wireless transmission range, hence practically all nodes may act as both hosts and routers. Mobile ad hoc networks require no centralized administration or fixed network infrastructure and can be quickly and inexpensively set up as needed. They can thus be used in scenarios where no infrastructure exists, such as military applications, emergent operations, personal electronic device networking, and civilian applications like an ad-hoc meeting or an ad-hoc classroom.

With more and more application, security for mobile ad hoc networks becomes increasingly important. A lot of secure solutions for mobile ad hoc networks have been proposed by far. But most of them are key management and authentication [1][2][3],

* He is a Ph.D. candidate at the department of Computing and Information Technology, Fudan University. His research interests are network security and mobile computing. Phn: +86-21-65643187, E-mail: pyi_edu@yahoo.com.cn

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 1183 – 1192, 2005.

secure routing protocol[4][5]. Most of those are prevention techniques. The prevention methods, such as encryption and authentication, used in mobile ad hoc networks can reduce attacks, but hardly eliminate them. When nodes roam in a hostile environment with relatively poor physical protection, they have probability of being compromised. The compromised nodes may launch attacks within the networks. Encryption and authentication can not defend against compromised nodes, which carry the private keys. In addition, a perfect protective security solution is for all practical purposes impossible and no matter how many intrusion prevention measures are inserted in a networks, there are always some weak links that one could exploit to break in. Intrusion detection should be the second wall of defence for security in mobile ad hoc networks.

This paper analyzes some of the vulnerabilities, specifically discussing attacks against DSR that manipulate the routing messages. We propose a solution based on finite state machines intrusion detection to detect attacks on DSR.

First, we design the distributed and cooperative intrusion detection architecture, which are composed of distribute monitor nodes. Intrusion detection in mobile ad hoc networks must be carried out in a distributed fashion because of the absence of infrastructure and the centre administration. Each network monitor node runs independently and monitors all nodes in its zone to find the local intrusions. To track some moving node, they may exchange information with neighbour monitors. Considering resource constraint, only some nodes in mobile ad hoc networks are selected as network monitors. We describe an algorithm which the nodes can periodically, randomly and fairly elect a monitor node for the entire zone.

Secondly, we propose a finite state machine (FSM)-based intrusion detection system which can detect attacks on the DSR routing protocol. In the FSM-based intrusion detection, the correct behaviours of critical objects are manually abstracted and crafted as security specifications, and this is compared with the actual behaviour of the objects. The technique may detect previously unknown attacks, while exhibiting a low false positive rate. Network monitors trace data flow on every node and audit every forwarding packet by finite state machines. If some node behaves in an incorrect manner, he will be found and some alarm will be sent out.

The rest of the paper is organized as follows. In section 2, we survey the related work in intrusion detection for mobile ad hoc networks. We introduce some background knowledge, such as Intrusion Detection System (IDS), Dynamic Source Routing and some attacks for DSR in section 3. In section 4, we describe the proposed approach. In section 5, we evaluate the effect of our intrusion detection approach. Finally, we discuss summarize the paper in section 6.

2 Related Work

Zhang and Lee describe a distributed and cooperative intrusion detection model [9]. In this model, an IDS agent runs at each mobile node, and performs local data collection and local detection, whereas cooperative detection and global intrusion response can be triggered when a node reports an anomaly. The main contribution of the paper is that it presents a distributed and cooperative intrusion detection architecture based on statistical anomaly detection techniques. However, the design of actual detection

techniques, their performance as well as verification were not addressed in the article. Zhang and Lee describe its experiments and performance in [6].Oleg Kachirski and Ratan Guha propose a distributed intrusion detection system based on mobile agent technology [7]. In contrast to the above architecture, the agents in [7] do not run on every node and they can be dynamically increased and decreased according to the resource of networks. Its architecture is aimed to minimize costs of network monitoring and maintaining a monolithic IDS system.

R. S. Puttini et al propose a distributed and modular architecture for IDS [10], and a signatures-based approach is propose to detect two kinds of intrusion. The architecture may not detect unknown attack. Yi-an Huang and Wenke Lee address a cooperative intrusion detection system for ad hoc networks [11]. In the paper, a set of rules is presented to identify the attack type or misbehaving nodes. But the author seems to ignore the attack of modification.

Bo Sun et al present a intrusion detection agent model which utilizes a Markov Chain based anomaly detection algorithm to construct the local detection engine [12]. P. Albers et al present a general intrusion detection architecture using agent [13]. In the architecture, the agents choose to use Simple Network Management Protocol (SNMP) data located in management information bases (MIB) as the audit source. S. Bhargava and D.P. Agrawal present the intrusion detection and intrusion response model for ad hoc networks [15]. Wang Weichao et al present the detection of false destination sequence numbers carried in RREQ packet [16].

3 Background

3.1 Overview of DSR [8]

The Dynamic Source Routing (DSR) is an entirely on-demand ad hoc network routing protocol, which composed of two parts: Route Discovery and Route Maintenance. In DSR, whenever a node needs to send a packet to some destination for which does not currently have a route to that destination in its Route Cache, the node initiates Route Discovery to find a route. The initiator broadcast a ROUTE REQUEST packet to its neighbours, specifying the target and a unique identifier from the initiator. Each node receiving the ROUTE REQUEST, if it has recently seen this request identifier from the initiator, discards the REQUEST. Otherwise, it appends its own node address to a list in the REQUEST and rebroadcasts the REQUEST. When the ROUTE REQUEST reaches its target node, the target sends a ROUTE REPLY back to the initiator of the REQUEST, including a copy of the accumulated list of addresses from the REQUEST. When the REPLY reaches the initiator of the REQUEST, it caches the new route in its Route Cache. The intermediate node also sends a ROUTE REPLY, if it has a route to the destination.

Route Maintenance is the mechanism by which a node sending a packet along a specified route to some destination detects if that route has broken. If, after a limited number of local retransmissions of the packet, a node in the route is unable to make this confirmation, it returns a ROUTE ERROR to the original source of the packet, identifying the link from itself to the next node as broken. The sender then removes this broken link from its Route Cache; for subsequent packets to this destination, the

sender may use any other route to that destination in its Cache, or it may attempt a new Route Discovery for that target if necessary.

3.2 Vulnerabilities and Attacks for DSR

DSR does not address security concerns, so it allows intruders to easily launch various kinds of attacks by modifying the route information. In DSR, some critical fields such as source address, destination address, address list, are very important and any misuse of these fields can cause DSR malfunction. An intruder may make use of the following ways against DSR.

- Impersonate a node S by forging a ROUTE REQUEST with its address as the originator address.
- When forwarding a ROUTE REQUEST, insert, delete and modify the address list.
- Impersonate a node D by forging a ROUTE REPLY with its address as a destination address.
- Selectively, not forward certain ROUTE REQUEST, ROUTE REPLY and data messages.
- Forge a ROUTE ERROR pretend it is the node in the route and send it to the initiator.

The above attacks can result in the following consequences.

- Blackhole: All traffic are redirected to a specific node, which may not forward any traffic at all.
- Routing Loop: A loop is introduced in a route path.
- Network Partition: A connected network is partitioned into k subnets where nodes in different subnets cannot communicate even though a route between them actually does exist.
- Sleep Deprivation: A node is forced to exhaust its battery power.
- Denial-of-Service: A node is prevented from receiving and sending data packets to its destinations.

4 FSM-Based Detection for DSR

4.1 Algorithm of Voting Monitor

The resources of battery power, CPU, memory in nodes are limited, and it is not efficient to make each node a monitor node. As a result, we may select some nodes as monitors to monitor the entire networks in order to save networks resource. The network monitor is the node which monitors the behavior of nodes within its monitor zone. The monitor zone is 1-hop vicinity of the monitor.

The process of voting monitor should is fairness and randomness. By fairness, we mean that every node should have a fair chance to serve as a monitor. Note that fairness has two components, fair election, and equal service time. We currently do not consider differentiated capability and preference and assume that every node is

equally eligible. Thus, fair election implies randomness in election decision, while equal service time can be implemented by periodical fair re-election. The randomness of the election process can guarantee the security. When some monitor node is compromised, it may not carry out the normal monitoring function and can launch certain attacks without being detected because it is the only node in the zone that is supposed to run the IDS and its IDS may have been disabled already. But after a service period, another node may be selected as monitor. At that time, the intrusion will be found by the normal monitor node.

The algorithm is composed of two parts, namely selection phase and maintain phase. In selection phase, the monitor is selected by competition. At first there is no monitor in networks. After a period, any node may broadcast the packet "I am monitor" and become a monitor. The packet can not be forwarded. Any node who receives the announcement becomes a monitored node and can not broadcast the announcement. When a monitor is selected, the selection phase is finished and go to the maintain phase. In maintain phase, the monitor broadcast the announcement periodically to keep up its monitor role. After a period, the monitor will terminate its monitor work and a new selection phase will begin. In order to insure fairness and randomness of selection, the predecessor can not take part in the process of selection, unless it is the only node in the entire zone.

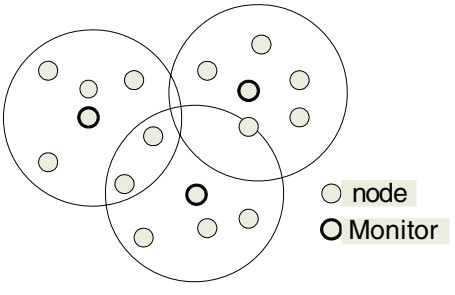


Fig. 1. monitor and its monitor zone

The monitors or nodes may move out of the zone due to dynamic topology. If any node does receive the announcement packet overtime, it can start to selection process and declare that itself is a monitor. Figure 1 shows monitors and their monitor zone.When two monitors move next to each other over an extended period of time, one whose ID is bigger will lose its role of monitor. As a result, whenever a monitor hears announcement messages from another monitor, it sets some time to expire. When expired, it will check if it is still in contention with the monitor, by checking if the monitor is still in its neighbor. If so, it compares its own ID with that of the other monitor's. The one with a smaller ID will continue to act as monitor. The one with a bigger ID gives up its role as monitor.

4.2 Finite State Machine Constraints

A monitor employs a finite state machine (FSM) for detecting incorrect behavior in a node. It maintains a FSM for each data flow in each node. In DSR, a node can receive

and forward four kinds of packets, i.e. ROUTE REQUEST, ROUTE REPLY, ROUTE ERROR and DATA. We firstly address how to deal with ROUTE REQUEST flow.

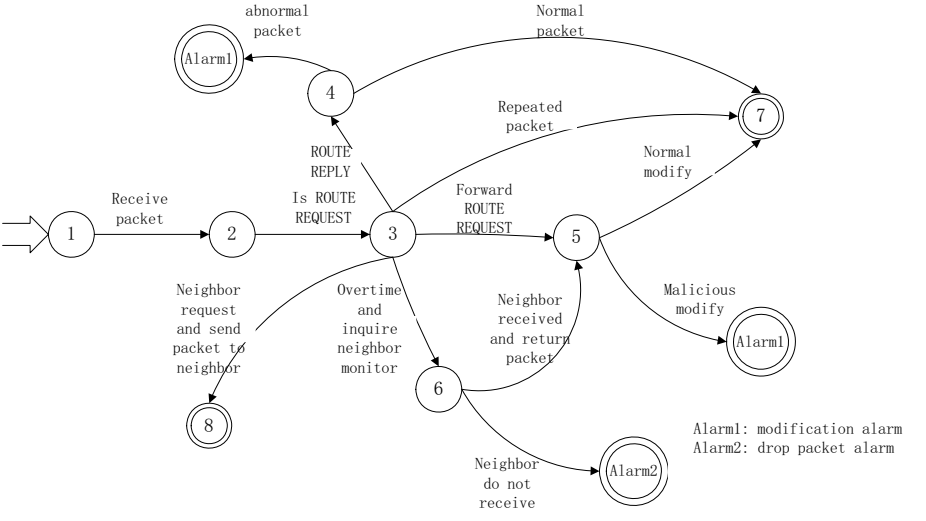


Fig. 2. The finite state machine constraints when received packet of ROUTE REQURE

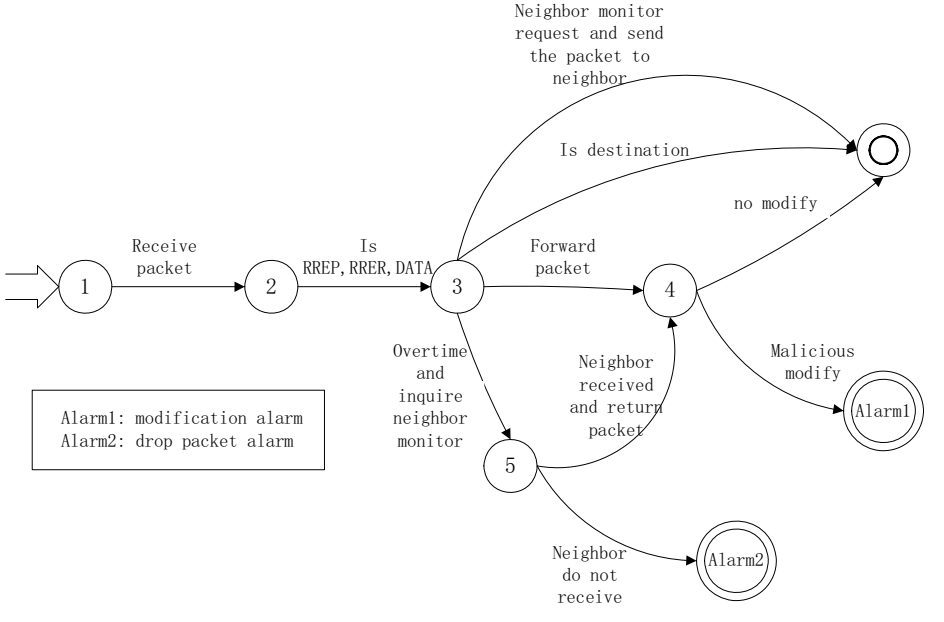


Fig. 3. The finite state machine constraints when received packet of ROUTE REPLY, ROUTE ERROR, DATA

Figure 2 shows the constraints of ROUTE REQUEST. The start state is 1. When the node receives a packet, FSM go to state 2. If the packet is ROUTE REQUEST, FSM go to state 3. If it is the target of the ROUTE REQUEST, the node returns a ROUTE REPLY to the initiator of the ROUTE REQUEST. FSM go to state 4 and check the packet of ROUTE REPLY according as routing protocol. If some fields of ROUTE REPLY are maliciously modified, FSM goes to state alarm1 and alert modification alarm, otherwise, FSM goes to terminal state 7. If this node has recently seen the same ROUTE REQUEST, it discards the packet and FSM goes to terminal state 7. If the node forwards the ROUTE REQUEST, FSM goes to state 5 and checks the forwarded packet according as routing protocol. If some fields of ROUTE REQUEST are maliciously modified, FSM goes to state alarm1 and alert modification alarm, otherwise, FSM goes to terminal state 7. If the packet has not been forwarded after a specified time, FSM goes to state 6. At that time the node may move out of the zone of the monitor and the monitor can not hear it forward the packet. Therefore the monitor inquire neighbor monitor whether it has forwarded the packet. If neighbor has received the packet, he will send it packet to the monitor for comparison. FSM goes to state 5. If no neighbor has received the packet, FSM goes to state Alarm2 and alert the alarm of drop packet. If neighbor monitor inquire about the packet, the monitor send the packet to neighbor to neighbor for comparison and FSM goes to terminal state 8.

We use the same FSM for three kinds of packets, i.e. ROUTE REPLY, ROUTE ERROR, DATA, for they are disposed at the same process. Figure 3 shows their FSM. The start state is 1. When the node receives a packet, FSM go to state 2. If the packet is one of the three packets, FSM go to state 3. If it is the target of the packet, FSM goes to terminal state. If neighbour monitor inquire the packet and the packet is sent to neighbour, FSM also goes to terminal state. If the node forwards the packet, FSM goes to state 4 and check the forwarded packet according as routing protocol. If some fields of the packet are maliciously modified, FSM goes to state alarm1 and alert modification alarm, otherwise, FSM goes to terminal state. When the node does not forward the packet within a period time, the monitor will inquire its neighbour monitors. If some neighbour received the packet, it will send it to the monitor for comparison and FSM goes to state 4. Otherwise FSM goes to state Alarm2.

Figure 2 and figure 3 show the process when a node receives a packet. Figure 4 shows the process when a node sends a packet. And the packet is not heard by the monitor, Otherwise the process is figure 2 or figure 3. The start state is 1. When the node receives a packet, FSM go to state 2. Then, if the packet is originated packet, FSM goes to state 6. The monitor compares the source address of packet with the address of the node which has sent the packet. If two addresses are marching, FSM goes to terminal state. Otherwise, FSM goes to state Alarm3 and alert impersonation alarm. It implies that the node is impersonating another node. If the packet is forwarded packet, FSM goes to state 3. In originated packet, the node address is source address and the node address is in address list in forwarded packet. Because the monitor does not see the packet, it inquires neighbour monitors for the packet. FSM goes to state 4. If no neighbour received the packet once, FSM goes to Alarm4, the node may fabricate a packet.

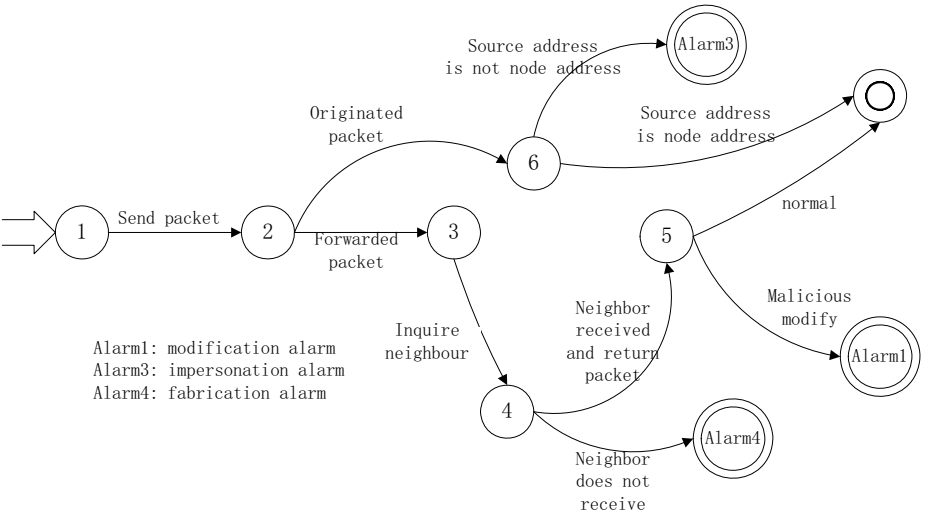


Fig. 4. The finite state machine constraints when node sends a packet

If some neighbour received the packet, it will send the packet to the monitor. FSM goes to state 5 and checks the forwarded packet according as routing protocol. If some fields of packet are maliciously modified, FSM goes to state alarm1 and alert modification alarm, otherwise, FSM goes to terminal state.

5 Experimental Results

To study the feasibility of our intrusion detection approach, we have implemented intrusion detection in a network simulator and conducted a series of experiments to evaluate its effectiveness. We used the wireless networks simulation software, from Network Simulator ns-2 [17]. It includes simulation for wireless ad-hoc network infrastructure, popular wireless ad-hoc routing protocols (DSR, DSDV, AODV and others), and mobility scenario and traffic pattern generation.

Our simulations are based on a 1500 by 300 meter flat space, scattered with 50 wireless nodes. The nodes move from a random starting point to a random destination with a speed that is randomly chosen. The speed is uniformly distributed between 0-20m/sec. As the destination is reached another random destination is targeted after a pause time. The MAC layer used for the simulations is IEEE 802.11, which is included in the ns-2. The transport protocol used for our simulations is User Datagram Protocol (UDP). Each data packet is 512 bytes long. The traffic files are generated such that the source and destination pairs are randomly spread over the entire network. The number of sources is 10 in the network. The scenario files determine the mobility of the nodes. The mobility model used random way point model in a rectangular field. Duration of the simulations is 900 seconds.

The simulations have been performed with malicious node created in the network and DSR protocol integrated with our intrusion detection model. By the analysis of

section 2, we simulate 4 types of attack. Attack 1 is illegal modification which the intruder illegally inserts, deletes, and modifies the address list when the intruder forwards a packet. Attack 2 is to drop packets which the intruder does not forward any packets and only receives packets. Attack 3 is impersonation which the intruder impersonate another node send some packets, such as ROUTE REQUEST, ROUTE REPLY and ROUTE ERROR. Attack 4 is fabrication which the intruder forges some packets which are not sent by the initiator. Table 1 show that detection rates and false alarms rates. The detection rate of attack 3 is highest. The main reason may be that the monitor directly compares the source address of packet with the address of the node which has sent the packet and the monitor need not the information from other monitor. The detection rate of attack 2 is lowest. The main reason may be that the monitor has to get information from the other monitor before it makes a judgment. We can draw a conclusion which our approach can detect the intrusion efficient with low false alarm rate from the simulation result.

Table 1. Detection performance

Attack type	Detection rate	False alarm rate
Attack 1	91.3%	2.9%
Attack 2	83.7%	5.7%
Attack 3	97.4%	1.3%
Attack 4	88.5%	7.2%

6 Conclusion

We propose a FSM-based intrusion detection system that can detect attacks on the DSR. In the system, firstly we propose an algorithm of selecting monitor for distributed monitoring all nodes in networks. Secondly, we manually abstract the correct behaviors of the node according as DSR and compose the finite state machine of node behavior. Intrusions, which usually cause node to behavior in an incorrect manner, can be detected without trained date or signature. Meanwhile, our IDS can detect unknown intrusion with fewer false alarms. As a result, we propose a distributed network monitor architecture which traces data flow on each node by means of finite state machine.

References

1. Lidong Zhou, Zygmunt J. Haas Securing ad hoc networks IEEE Networks Special Issue on Network Security November/December, 1999
2. Srdjan Capkun, Levente Nuttyan, Jean-Pierre Hubaux, Self-organized public-key Management for mobile ad hoc networks, IEEE Transactions on mobile computing, Vol.2, No.1, January-March, 2003
3. Aldar Chan, Distributed Symmetric Key Management for Mobile Ad hoc Networks, IEEE INFOCOM'04, Hong Kong, March 2004

4. Yih-Chun Hu, David B. Johnson, and Adrian Perrig SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks in Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002), pp. 3-13, IEEE, Calicoon, NY, June 2002
5. Yih-Chun Hu, Adrian Perrig, David B. Johnson. Ariadne: A secure On-Demand Routing Protocol for Ad hoc Networks in Proceedings of the MobiCom 2002, September 23-28, 2002, Atlanta, Georgia, USA
6. Yongguang Zhang & Wenke Lee, Intrusion Detection Techniques for Mobile Wireless Networks, Mobile Networks and Applications, 2003
7. Oleg Kachirski, Ratan Guha, Intrusion Detection Using Mobile Agents in Wireless Ad Hoc Networks, IEEE Workshop on Knowledge Media Networking (KMN'02)
8. David B. Johnson, David A. Maltz, Yih-Chun Hu, The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR), Internet-Draft, draft-ietf-manet-dsr-09.txt, 15 April 2003, http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-09.txt
9. Yongguang Zhang & Wenke Lee, Intrusion Detection in Wireless Ad-Hoc Networks in Proceedings of The Sixth International Conference on Mobile Computing and Networking (MobiCom 2000), Boston, MA, August 2000
10. R. S. Puttini, J-M. Percher, L. Mé, O. Camp, R. de Sousa Jr., C. J. Barenco Abbas, L. J. Garcia Villalba. A Modular Architecture for Distributed IDS in MANET, Proceedings of the 2003 International Conference on Computational Science and Its Applications (ICCSA 2003), Springer Verlag, LNCS 2668, San Diego, USA, 2003
11. Yi-an Huang, Wenke Lee, A Cooperative Intrusion Detection System for Ad Hoc Networks, 2003 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '03) , Fairfax, VA, USA, October 31, 2003
12. B. Sun, K. Wu, and U.W. Pooch, Routing Anomaly Detection in Mobile Ad Hoc Networks, Proceedings of 12th International Conference on Computer Communications and Networks (ICCCN 03), Dallas, Texas, October 2003, pp. 25-31
13. P. Albers, O. Camp, J.-M. Percher, B. Jouga, L. Mé, and R. Puttini. Security in Ad Hoc Networks: a General Intrusion Detection Architecture Enhancing Trust Based Approaches. In Proceedings of the First International Workshop on Wireless Information Systems (WIS-2002), Apr. 2002
14. Denning D.E. An intrusion detection model, IEEE Transactions on Soft- ware Engineering, SE-13:222–232, 1987
15. S. Bhargava and D.P. Agrawal, Security Enhancements in AODV Protocol for Wireless Ad Hoc Networks, Vehicular Technology Conference, 2001, vol. 4, pp. 2143-2147
16. Weichao Wang, Yi Lu, Bharat, K. Bhargava, On Vulnerability and Protection of Ad Hoc On-demand Distance Vector Protocol, in Proceedings of 10th IEEE International Conference on Telecommunication (ICT), 2003. 16.
17. http://www.isi.edu/nsnam/ns/

Author Index