Eduardo César Michael Alexander
Achim Streit Jesper Larsson Träff
Christophe Cérin Andreas Knüpfer
Dieter Kranzlmüller Shantenu Jha (Eds.)

LNCS 5415

Euro-Par 2008 Workshops – Parallel Processing

VHPC 2008, UNICORE 2008, HPPC 2008, SGS 2008,
PROPER 2008, ROIA 2008, and DPA 2008
Las Palmas de Gran Canaria, Spain, August 2008
Revised Selected Papers

Lecture Notes in Computer Science 5415

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

Editorial Board

David Hutchison
Lancaster University, UK
Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA
Josef Kittler
University of Surrey, Guildford, UK
Jon M. Kleinberg
Cornell University, Ithaca, NY, USA
Alfred Kobsa
University of California, Irvine, CA, USA
Friedemann Mattern
ETH Zurich, Switzerland
John C. Mitchell
Stanford University, CA, USA
Moni Naor
Weizmann Institute of Science, Rehovot, Israel
Oscar Nierstrasz
University of Bern, Switzerland
C. Pandu Rangan
Indian Institute of Technology, Madras, India
Bernhard Steffen
University of Dortmund, Germany
Madhu Sudan
Massachusetts Institute of Technology, MA, USA
Demetri Terzopoulos
University of California, Los Angeles, CA, USA
Doug Tygar
University of California, Berkeley, CA, USA
Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Eduardo César Michael Alexander
Achim Streit Jesper Larsson Träff
Christophe Cérin Andreas Knüpfer
Dieter Kranzlmüller Shantenu Jha (Eds.)

Euro-Par 2008 Workshops – Parallel Processing

VHPC 2008, UNICORE 2008, HPPC 2008, SGS 2008,
PROPER 2008, ROIA 2008, and DPA 2008
Las Palmas de Gran Canaria, Spain, August 25-26, 2008
Revised Selected Papers

Volume Editors

Eduardo César
Universidad Autónoma de Barcelona, Spain
E-mail: eduardo.cesar@uab.es

Michael Alexander
Wirtschaftsuniversität Wien, Austria
E-mail: malexand@wu-wien.ac.at

Achim Streit
Jülich Supercomputing Centre, Germany
E-mail: a.streit@fz-juelich.de

Jesper Larsson Träff
NEC Laboratories Europe, Sankt Augustin, Germany
E-mail: traff@it.neclab.eu

Christophe Cérin
Université de Paris Nord, LIPN, France
E-mail: christophe.cerin@lipn.univ-paris13.fr

Andreas Knüpfer
Technische Universität Dresden, Germany
E-mail: andreas.knupfer@tu-dresden.de

Dieter Kranzlmüller
LMU München, Germany
E-mail: dk@gup.jku.at

Shantenu Jha
Louisiana State University, USA
E-mail: sjha@cct.lsu.edu

Library of Congress Control Number: Applied for

CR Subject Classification (1998): C.1-4, D.1-4, F.1-3, G.1-2, H.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-00954-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-00954-9 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2009
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12643462 06/3180 5 4 3 2 1 0

Preface

Parallel and distributed processing, although within the focus of computer science research for a long time, is gaining more and more importance in a wide spectrum of applications. These proceedings aim to demonstrate the use of parallel and distributed processing concepts in different application fields, and attempt to spark interest in novel research directions to parallel and high-performance computing research in general.

The objective of these workshops is to specifically address researchers coming from university, industry and governmental research organizations and application-oriented companies in order to close the gap between purely scientific research and the applicability of the research ideas to real-life problems.

Euro-Par is an annual series of international conferences dedicated to the promotion and advancement of all aspects of parallel and distributed computing.

The 2008 event was the 14th issue of the conference. Euro-Par has for a long time been eager to attract colocated events sharing the same goal of promoting the development of parallel and distributed computing, both as an industrial technique and an academic discipline, extending the frontier of both the state of the art and the state of the practice. Since 2006, Euro-Par has been offering researchers the chance to colocate advanced technical workshops back-to-back with the main conference. This is for a mutual benefit: the workshops can take advantage of all technical and social facilities that are set up for the conference, so that the organizational tasks are kept to a minimal level; the conference can rely on workshops to experiment with specific areas of research that are not yet mature enough, or too specific, to lead to an official, full-fledged topic at the conference.

The 2006 and 2007 events were quite successful, and were extended to a larger size in 2008, where nine events were colocated with the main Euro-Par Conference:

- **CoreGRID Symposium** is the major annual event of the CoreGRID European Research Network on Foundations, Software Infrastructures and Applications for large-scale distributed, grid and peer-to-peer technologies. It is also an opportunity for a number of CoreGRID Working Groups to organize their regular meetings. The proceedings have been published in a specific volume of the Springer CoreGRID series, *Towards Next Generation Grids.*
- **GECON 2008** is the 5th International Workshop on Grid Economic and Business Model. Euro-Par was eager to attract an event about this very important aspect of grid computing, which has often been overlooked by scientific researchers of the field. Its proceedings are published in a separate volume of Springer's *Lecture Notes in Computer Science* series.
- **VHPC 2008** is the Workshop on Virtualization/Xen in High-Performance Cluster and Grid Computing. Virtual machine monitors (VMMs) are now

integrated with a variety of operating systems and are moving out of research labs into scientific, educational and operational usage. This workshop aimed to bring together researchers and practitioners active in exploring the application of virtualization in distributed and high-performance cluster and grid computing environments. This was a unique opportunity for the Euro-Par community to make connections with this very active research domain.

- **UNICORE Summit 2008** brought together researchers and practitioners working with UNICORE in the areas of grid and distributed computing, to exchange and share their experiences, new ideas and latest research results on all aspects of UNICORE. The UNICORE grid technology provides a seamless, secure and intuitive access to distributed grid resources.
  This was the fourth meeting of the UNICORE community, after a meeting in Sophia-Antipolis, France, in 2005, and a colocated meeting at Euro-Par 2006 in Dresden, Germany, in 2006, and Euro-Par 2007 in Rennes, France.
- **HPPC 2008** is the Second Workshop on Highly Parallel Processing on a Chip. With a number of both general and special purpose multi-core processors already on the market, it is foreseeable that new designs with a substantial number of processing cores will emerge to meet demands for extremely high performance, dependability and controllable power consumption in mobile and embedded devices, and in response to the convergence of communication, media and compute devices. The HPPC workshop aims to be(come) a forum for discussion of the major challenges to architecture, language and compiler design, algorithms and application developments, in order to fully (or acceptably) exploit the raw compute power of multi-core processors with a significant amount of parallelism.
- **SGS 2008** is the First Workshop on Secure, Trusted, Manageable and Controllable Grid Services. It refers to the notions of security, the way we manage such large systems and the way we control the grid system. For instance, the word 'controllable' means: how we measure the activity of the grid and how we report it. The word 'manageable' means: 'how we deploy the grid architecture, the grid softwares, and how we start jobs (under controllable events such as the availability of resources). The word 'security' refers to the traditional fields of authentication, fault tolerance but refers also to safe execution (how to certify results, how to adapt computation according to some metric). Moreover, all these services should collaborate making the building of middleware a challenging problem. The building of chains of trust between software components as well as the integration of security and privacy mechanisms across multiple autonomous and/or heterogeneous grid platforms are key challenges for the community.
- **The PROPER 2008** workshop was organized on behalf of the Virtual Institute for High Productivity Supercomputing (VI-HPS), which aims at improving the quality and accelerating the development process of complex simulation codes in science and engineering that are being designed to run on highly parallel computer systems. One part of this mission is the development of integrated state-of-the-art programming tools for high-performance computing that assist

programmers in diagnosing programming errors and optimizing the performance of their applications.

Accordingly, the workshop topics cover tools for parallel program development and analysis as well as general performance measurement and evaluation approaches. Last but not least, it includes success stories about optimization or parallel scalability achieved using the tools. In particular, the workshop wants to stimulate discussion between tool developers and experts on one hand and tool users and application developers on the other hand. Furthermore, it especially supports younger researchers to present their work.

- **ROIA 2008** is the First International Workshop on Real-Time Online Interactive Applications on the Grid. It aimed to bring together researchers from the domain of ROIAs and grid computing in order to exchange knowledge, experiences, ideas and concepts for combining both fields. The event was closely related to the research perfomed in the European edutain@grid project.
- **DPA 2008** aimed to determine where programming abstractions are important and where non-programmatic abstractions are likely to make greater impact in enabling applications to effectively utilize distributed infrastructure. This workshop will have a balance of applications and topical infrastructure developments (such as abstractions for Clouds).

The reader will find in this volume the proceedings of the last seven events.

Hosting Euro-Par 2008 and these colocated events in Las Palmas de Gran Canaria would not have been possible without the support and the help of different institutions and numerous people.

Although we are thankful to many more people, we are particularly grateful to the workshop organizers: Martti Forsell and Jesper Larsson Träff for HPPC 2008; Achim Streit and Wolfgang Ziegler for UNICORE Summit 2008; and Michael Alexander and Stephen Childs for VHPC 2008. It has been a pleasure to collaborate with them on this project.

We particularly thank them for their interest in our proposal and their trust and availability along the entire preparation process.

Euro-Par 2008 was hosted on the university campus and we would like to thank the University Institute for Intelligent Systems and Numerical Applications in Engineering of the Universidad de Las Palmas de Gran Canaria for the support and infrastructure. We gratefully acknowledge the great organizational support of the Computer Architecture and Operating Systems Department of the Universidad Autónoma de Barcelona. We would also like to thank the Cabildo de Gran Canaria and the City Council of Las Palmas de Gran Canaria for they institutional support.

Finally, we are grateful to Springer for agreeing to publish the proceedings of these seven workshops in a specific volume of its *Lecture Notes in Computer Science* series. We are definitely eager to pursue this collaboration.

It has been a great pleasure to work together on this project in Las Palmas de Gran Canaria.

We hope that the current proceedings are beneficial for the sustainable growth and awareness of parallel and distributed computing concepts in future applications.

December 2008

Eduardo César
Michael Alexander
Achim Streit
Jesper Larsson Träff
Christophe Cérin
Andreas Knüpfer
Dieter Kranzlmüller
Shantenu Jha

Organization

Euro-Par Steering Committee

Chair

Christian Lengauer	University of Passau, Germany

Vice-Chair

Luc Bougé	ENS Cachan, France

European Representatives

José Cunha	New University of Lisbon, Portugal
Marco Danelutto	University of Pisa, Italy
Rainer Feldmann	University of Paderborn, Germany
Christos Kaklamanis	Computer Technology Institute, Greece
Anne-Marie Kermarrec	IRISA, Rennes, France
Paul Kelly	Imperial College, UK
Harald Kosch	University of Klagenfurt, Austria
Thomas Ludwig	University of Heidelberg, Germany
Emilio Luque	University Autonoma of Barcelona, Spain
Luc Moreau	University of Southampton, UK
Wolfgang Nagel	Dresden University of Technology, Germany
Rizos Sakellariou	University of Manchester, UK

Non-European Representatives

Jack Dongarra	University of Tennessee at Knoxville, USA
Shinji Tomita	Kyoto University, Japan

Honorary Members

Ron Perrott	Queen's University Belfast, UK
Karl Dieter Reinartz	University of Erlangen-Nuremberg, Germany

Observers

Domingo Benitez	University of Las Palmas, Gran Canaria, Spain
Henk Sips	Delft University of Technology, The Netherlands

Euro-Par 2008 Local Organization

Conference Co-chairs

Emilio Luque	UAB General Chair
Domingo Benítez	ULPGC Vice-Chair
Tomàs Margalef	UAB Vice-Chair

Local Organizing Committee

Eduardo César (UAB)
Ana Cortés (UAB)
Daniel Franco (UAB)
Elisa Heymann (UAB)
Anna Morajko (UAB)
Juan Carlos Moure (UAB)
Dolores Rexachs (UAB)
Miquel Àngel Senar (UAB)
Joan Sorribes (UAB)
Remo Suppi (UAB)

Web and Technical Support

Daniel Ruiz (UAB)
Javier Navarro (UAB)

Euro-Par 2008 Workshop Program Committees

Third Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC 2008)

Program Chairs

Michael Alexander (Chair)	WU Vienna, Austria
Stephen Childs (Co-chair)	Trinity College, Dublin, Ireland

Program Committee

Jussara Almeida	Federal University of Minas Gerais, Brazil
Padmashree Apparao	Intel Corp., USA
Hassan Barada	Etisalat University College, UAE
Volker Buege	University of Karlsruhe, Germany
Simon Crosby	Xensource, UK
Marcus Hardt	Forschungszentrum Karlsruhe, Germany
Sverre Jarp	CERN, Switzerland
Krishna Kant	Intel Corporation, USA
Yves Kemp	University of Karlsruhe, Germany

Naoya Maruyama	Tokyo Institute of Technology, Japan
Jean-Marc Menaud	Ecole des Mines de Nantes, France
José E. Moreira	IBM T.J. Watson Research Center, USA
Jose Renato Santos	HP Labs, USA
Andreas Schabus	Microsoft , Austria
Yoshio Turner	HP Labs, USA
Andreas Unterkircher	CERN, Switzerland
Dongyan Xu,	Purdue University, USA

UNICORE Summit 2008

Program Chairs

Achim Streit	Forschungszentrum Jülich, Germany
Wolfgang Ziegler	Fraunhofer Gesellschaft SCAI, Germany

Program Committee

Agnes Ansari	CNRS-IDRIS, France
Rosa Badia	Barcelona Supercomputing Center, Spain
Thomas Fahringer	University of Innsbruck, Austria
Donal Fellows	University of Manchester, UK
Anton Frank	LRZ Munich, Germany
Edgar Gabriel	University of Houston, USA
Alfred Geiger	T-Systems, Germany
Fredrik Hedman	KTH-PDC, Sweden
Odej Kao	Technical University of Berlin, Germany
Paolo Malfetti	CINECA, Italy
Ralf Ratering	Intel GmbH, Germany
Mathilde Romberg	Forschungszentrum Jülich, Germany
Bernd Schuller	Forschungszentrum Jülich, Germany
David Snelling	Fujitsu Laboratories of Europe, UK
Thomas Soddemann	Max-Planck-Institut für Plasmaphysik - RZG, Germany
Stefan Wesner	University of Stuttgart - HLRS, Germany
Ramin Yahyapour	University of Dortmund, Germany

Additional Reviewers

Max Berger
Kassian Plankensteiner

Third Workshop on Highly Parallel Processing on a Chip (HPPC 2008)

Program Chairs

Martti Forsell	VTT, Finland
Jesper Larsson Träff	NEC Laboratories Europe, NEC Europe Ltd., Germany

Program Committee

David Bader	Georgia Institute of Technology, USA
Gianfranco Bilardi	University of Padova, Italy
Martti Forsell	VTT, Finland
Anwar Ghuloum	Intel, USA
Peter Hofstee	IBM, USA
Chris Jesshope	University of Amsterdam, The Netherlands
Ben Juurlink	Technical University of Delft, The Netherlands
Darren Kerbyson	Los Alamos National Laboratory, USA
Christoph Kessler	University of Linköping, Sweden
Dominique Lavenier	IRISA - CNRS, France
Lasse Natvig	NTNU, Norway
Andrea Pietracaprina	University of Padova, Italy
Jesper Larsson Träff	NEC Laboratories Europe, NEC Europe Ltd., Germany
Uzi Vishkin	University of Maryland, USA

Workshop on Secure, Trusted, Manageable and Controllable Grid Services (SGS 2008)

Steering Committee

Pascal Bouvry	University of Luxembourg, Luxembourg
Christophe Cérin	University of Paris 13, France
Noria Foukia	Otago University, New Zealand
Jean-Luc Gaudiot	University of California, Irvine, USA
Mohamed Jemni	ESSTT, Tunisia
Kuan-Ching Li	Providence University, Taiwan
Jean-Louis Pazat	IRISA, France
Helmut Reiser	Leibniz Supercomputing Centre, Garching, Germany

Workshop on Productivity and Performance (PROPER 2008)

Program Chairs

Matthias S. Müller (Chair)
Andreas Knüpfer (Local Chair)

Program Committee

Matthias Müller	Technical University of Dresden (Chair)
Karl Füerlinger	University of Tennessee
Andreas Knüpfer	Technical University of Dresden
Bettina Krammer	University of Stuttgart
Allen Malony	University of Oregon
Dieter an Mey	RWTH Aachen University
Shirley Moore	University of Tennessee
Martin Schulz	Lawrence Livermore National Lab
Felix Wolf	Forschungszentrum Jülich

Real-Time Online Interactive Applications (ROIA) on the GRID

Program Chairs

Christoph Anthes
Thomas Fahringer
Dieter Kranzlmüller

Program Committee

Alexis Aragon	Darkworks S.A., France
Damjan Ceric	Amis d.o.o, Slovenia
Justin Ferris	IT Innovation Centre, University of Southampton,UK
Frank Glinka	Institute of Computer Science, University of Münster, Germany
Sergei Gorlatch	Institute of Computer Science, University of Münster, Germany
Alexandru Iosup	Parallel and Distributed Systems (PDS) Group, TU Delft, The Netherlands
Roland Landertshamer	Institute of Graphics and Parallel Processing, Joh. Kepler University Linz, Austria
Mark Lidstone	BMT Cordah Ltd., UK
Arton Lipaj	Amis d.o.o, Slovenia
Jens Müller-Iden	Institute of Computer Science, University of Münster, Germany
Vlad Nae	Institute for Computer Science, University of Innsbruck, Austria

Alexander Ploss	Institute of Computer Science, University of Münster, Germany
Radu Prodan	Institute for Computer Science, University of Innsbruck, Austria
Christopher Rawlings	BMT Cordah Ltd., UK
Mike Surridge	IT Innovation Centre, University of outhampton, UK
Jens Volkert	Institute of Graphics and Parallel Processing, Joh. Kepler University Linz, Austria

Abstractions for Distributed Systems (DPA 2008)

Program Chair

Shantenu Jha (LSU and eSI), Chair

Program Committee

Shantenu Jha (LSU and eSI)
Dan Katz (LSU)
Manish Parashar (Rutgers)
Omer Rana (Cardiff)
Murray Cole (Edinburgh)

Table of Contents

Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC 2008)

Preface . . . 1
Michael Alexander and Stephen Childs (Program Chairs)

Tools and Techniques for Managing Virtual Machine Images . . . 3
Håvard K.F. Bjerke, Dimitar Shiyachki, Andreas Unterkircher, and Irfan Habib

Dynamic on Demand Virtual Clusters in Grid . . . 13
Mario Leandro Bertogna, Eduardo Grosclaude, Marcelo Naiouf, Armando De Giusti, and Emilio Luque

Dynamic Provisioning of Virtual Clusters for Grid Computing . . . 23
Manuel Rodríguez, Daniel Tapiador, Javier Fontán, Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente

Dynamic Resources Management of Virtual Appliances on a Computational Cluster . . . 33
Alexander A. Moskovsky, Artem Y. Pervin, and Bruce J. Walker

Complementarity between Virtualization and Single System Image Technologies . . . 43
Jérôme Gallard, Geoffroy Vallée, Adrien Lèbre, Christine Morin, Pascal Gallard, and Stephen L. Scott

Efficient Shared Memory Message Passing for Inter-VM Communications . . . 53
François Diakhaté, Marc Perache, Raymond Namyst, and Herve Jourdren

An Analysis of HPC Benchmarks in Virtual Machine Environments . . . 63
Anand Tikotekar, Geoffroy Vallée, Thomas Naughton, Hong Ong, Christian Engelmann, and Stephen L. Scott

UNICORE Summit 2008

Preface . . . 73
Achim Streit and Wolfgang Ziegler (Program Chairs)

Space-Based Approach to High-Throughput Computations in UNICORE 6 Grids . . . 75
Bernd Schuller and Miriam Schumacher

The Chemomentum Data Services – A Flexible Solution for Data Handling in UNICORE 84
Katharina Rasch, Robert Schöne, Vitaliy Ostropytskyy, Hartmut Mix, and Mathilde Romberg

A Reliable and Fast Data Transfer for Grid Systems Using a Dynamic Firewall Configuration 94
T. Oistrez, E. Grünter, M. Meier, and R. Niederberger

Workflow Service Extensions for UNICORE 6 – Utilising a Standard WS-BPEL Engine for Grid Service Orchestration 103
S. Gudenkauf, W. Hasselbring, A. Höing, G. Scherp, and O. Kao

Benchmarking of Integrated OGSA-BES with the Grid Middleware 113
Fredrik Hedman, Morris Riedel, Phillip Mucci, Gilbert Netzer, Ali Gholami, M. Shahbaz Memon, A. Shiraz Memon, and Zeeshan A. Shah

Second Workshop on Highly Parallel Processing on a Chip (HPPC 2008)

Preface 123
Martti Forsell and Jesper Larsson Träff (Program Chairs)

Models for Parallel and Hierarchical On-Chip Computation (Abstract) 127
Gianfranco Bilardi

Building a Concurrency and Resource Allocation Model into a Processor's ISA (Abstract) 129
Chris Jesshope

Optimized Pipelined Parallel Merge Sort on the Cell BE 131
Jörg Keller and Christoph W. Kessler

Towards an Intelligent Environment for Programming Multi-core Computing Systems 141
Sabri Pllana, Siegfried Benkner, Eduard Mehofer, Lasse Natvig, and Fatos Xhafa

Adaptive Read Validation in Time-Based Software Transactional Memory 152
Ehsan Atoofian, Amirali Baniasadi, and Yvonne Coady

Compile-Time and Run-Time Issues in an Auto-Parallelisation System for the Cell BE Processor 163
Alastair F. Donaldson, Paul Keir, and Anton Lokhmotov

A Unified Runtime System for Heterogeneous Multi-core Architectures 174
Cédric Augonnet and Raymond Namyst

(When) Will CMPs Hit the Power Wall? 184
Cor Meenderinck and Ben Juurlink

Workshop on Secure, Trusted, Manageable and Controllable Grid Services (SGS 2008)

Preface 195
Christophe Cérin

Meta-Brokering Solutions for Expanding Grid Middleware Limitations 199
Attila Kertész, Ivan Rodero, and Francesc Guim

Building Secure Resources to Ensure Safe Computations in Distributed and Potentially Corrupted Environments 211
Sebastien Varrette, Jean-Louis Roch, Guillaume Duc, and Ronan Keryell

Simbatch: An API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems 223
Y. Caniou and J.-S. Gay

Analysis of Peer-to-Peer Protocols Performance for Establishing a Decentralized Desktop Grid Middleware 235
Heithem Abbes and Jean-Christophe Dubacq

Towards a Security Model to Bridge Internet Desktop Grids and Service Grids 247
Gabriel Caillat, Oleg Lodygensky, Etienne Urbah, Gilles Fedak, and Haiwu He

Workshop on Productivity and Performance (PROPER 2008)

Preface 261
Matthias Müller and Andreas Knüpfer (Program Chairs)

Enabling Data Structure Oriented Performance Analysis with Hardware Performance Counter Support 263
Karl Fürlinger, Dan Terpstra, Haihang You, Phil Mucci, and Shirley Moore

Complete Def-Use Analysis in Recursive Programs with Dynamic Data Structures 273
R. Castillo, F. Corbera, A. Navarro, R. Asenjo, and E.L. Zapata

Parametric Studies in Eclipse with TAU and PerfExplorer 283
Kevin A. Huck, Wyatt Spear, Allen D. Malony, Sameer Shende, and Alan Morris

Trace-Based Analysis and Optimization for the Semtex CFD Application – Hidden Remote Memory Accesses and I/O Performance 295
Holger Mickler, Andreas Knüpfer, Michael Kluge, Matthias S. Müller, and Wolfgang E. Nagel

Scalasca Parallel Performance Analyses of PEPC 305
Zoltán Szebenyi, Brian J.N. Wylie, and Felix Wolf

Comparing the Usability of Performance Analysis Tools 315
Christian Iwainsky and Dieter an Mey

Real-Time Online Interactive Applications on the Grid (ROIA 2008)

Preface 327
Christoph Anthes, Thomas Fahringer, and Dieter Kranzlmüller (Program Chairs)

Real-Time Performance Support for Complex Grid Applications 329
Marian Bubak, Włodzimierz Funika, Bartosz Baliś, Tomasz Szepieniec, Krzysztof Guzy, and Roland Wismüller

CoUniverse: Framework for Building Self-organizing Collaborative Environments Using Extreme-Bandwidth Media Applications 339
Miloš Liška and Petr Holub

Developing VR Applications for the Grid 352
Christoph Anthes, Roland Landertshamer, Helmut Bressler, and Jens Volkert

An Information System for Real-Time Online Interactive Applications 361
Vlad Nae, Jordan Herbert, Radu Prodan, and Thomas Fahringer

Securing Real-Time On-Line Interactive Applications in edutain@grid 371
J. Ferris, M. Surridge, and F. Glinka

The edutain@grid Portals – Providing User Interfaces for Different Kinds of Actors 382
Roland Landertshamer, Christoph Anthes, Jens Volkert, Bassem I. Nasser, and Mike Surridge

A Case Study on Using RTF for Developing Multi-player Online Games 390
Alexander Ploss, Frank Glinka, and Sergei Gorlatch

Abstractions for Distributed Systems (DPA 2008)

Preface 401
Shantenu Jha, Dan Katz, Manish Parashar, Omer Rana, and Murray Cole (Program Committee)

Co-design of Distributed Systems Using Skeleton and Autonomic Management Abstractions 403
M. Aldinucci, M. Danelutto, and P. Kilpatrick

Distributed Data Mining Tasks and Patterns as Services 415
Domenico Talia

ProActive Parallel Suite: From Active Objects-Skeletons-Components to Environment and Deployment 423
Denis Caromel and Mario Leyton

On Abstractions of Software Component Models for Scientific Applications 438
Julien Bigot, Hinde Lilia Bouziane, Christian Pérez, and Thierry Priol

Group Abstractions for Organizing Dynamic Distributed Systems 450
José C. Cunha, Carmen P. Morgado, and Jorge F. Custódio

Author Index 461

Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC 2008)

Virtual machine monitors (VMMs) are now integrated with a variety of operating systems and are moving out of research labs into scientific, educational and operational usage. Modern hypervisors exhibit a low overhead and allow concurrent execution of large numbers of virtual machines, each strongly encapsulated. VMMs can offer a network-wide abstraction layer of individual machine resources, thereby opening up new models for implementing high-performance computing (HPC) architectures in both cluster and grid environments. This workshop aims to bring together researchers and practitioners active in exploring the application of virtualization in distributed and high-performance cluster and grid computing environments.

Areas that are covered in the workshop series include VMM performance, VMM architecture and implementation, cluster and grid VMM applications, management of VM-based computing resources, hardware support for virtualization, but it is open to a wider range of topics.

As basic virtualization technologies mature, the main focus of research now is techniques for managing virtual machines in large-scale installations. This was reflected in this year's workshop, where five presentations were given on the management of virtualized HPC systems. In total seven papers were accepted for this year's workshop, with an acceptance rate of approximately 39%.

An invited talk by Bernhard Schott of the company Platform gave an overview of the company's products relative to virtualization.

The Chairs would like to thank the Euro-Par organizers, the members of the Program Committee along with the speakers and attendees, whose interaction created a stimulating environment. Our special thanks to Bernhardt Schott for accepting our invitation to speak at the workshop and we acknowledge the financial support of Citrix. VHPC is planning to continue the successful co-location with Euro-Par in 2009.

December 2008

Michael Alexander
Stephen Childs

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, p. 1, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Tools and Techniques for Managing Virtual Machine Images

Håvard K. F. Bjerke[1], Dimitar Shiyachki[1], Andreas Unterkircher[1], and Irfan Habib[2]

[1] CERN, 1211 Geneva 23, Switzerland
[2] Centre for Complex Cooperative Systems (CCCS), University of the West of England (UWE) Frenchay, Bristol BS16 1QY United Kingdom

Abstract. Virtual machines can have many different deployment scenarios and therefore may require generation of multiple VM images. OS Farm is a service that aims to provide VM images that are tailored and generated on the fly. In order to optimize generation of images, a layered copy-on-write image structure is used, and an image cache ensures that identical images are not regenerated.

Images can be several hundreds of megabytes large and thus can congest the network and delay their transfer. Content-Based Transfer is a technique which transfers only the difference between the source image and existing target client image data. We present an implementation which achieves an observed bandwidth close to the theoretical maximum and a significant reduction in network congestion.

1 Introduction

Virtualization can add agility to datacenters by providing flexible testing environments, failover with live-migration and satisfying different OS flavour requirements with consolidation. In all these scenarios it is important to have an infrastructure that efficiently handles the needed VM images. Sect. 2 presents a real scenario for the application of image management techniques, as a motivation for this work.

Libfsimage is a library and a standalone application, which generates VM images with a rich selection of Linux distributions. It is presented in Sect. 3.

OS Farm is a software application that aims to provide a user interface for generating and managing VM images. For generating images, it uses Libfsimage. It employs some techniques in order to optimize the generation and propagation of images, as described in Sect. 4.

The large sizes of VM images is a hurdle for managing images, particularly in image propagation. An implementation of the Content-Based Transfer technique, which optimizes the propagation of VM images over the network, is presented in Sect. 5.

This paper presents our work as a solution for image management with a good degree of configuration flexibility and performance.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 3–12, 2009.
© Springer-Verlag Berlin Heidelberg 2009

2 Application of Image Management Techniques in the EGEE/WLCG Grid

The Enabling Grids for E-scienceE (EGEE)[1] project, funded by the European Union, provides a seamless Grid infrastructure for e-Science. EGEE produces the gLite[2] middleware for grid computing. Tightly coupled to EGEE is the Worldwide LHC Computing Grid (WLCG)[3]. Its mission is to build and maintain a data storage and analysis infrastructure for the entire high energy physics community that will use the Large Hadron Collider (LHC), which is currently being built at the European Organization for Nuclear Research (CERN)[4].

2.1 Grid Middleware Certification

A section in CERN's IT Department is responsible for the integration, testing and release of the gLite middleware. This activity is carried out in collaboration with several partners all over Europe within EGEE. Testing gLite faces the problem that its components are under active development. To enable progress in certification the turnaround time from feature submission to certified state must be as small as possible.

Bug fixes and new features enter gLite via the concept of a patch. A grid testbed is being operated so that new patches can be applied to the relevant grid nodes. However, certification of several patches at the same time can cause conflicts on the testbed. A non functional patch may spoil the whole testbed. To cope with such problems an infrastructure of virtual machines based on Xen was established so that certifiers can bring up grid nodes with a certain patch independently.

GLite is available on different Linux flavors and architectures: Scientific Linux CERN[5] 3 and 4 on i686 and x86_64 platforms. More Linux flavors (e.g. Debian) are under development. As all these combinations are found in production, interactions of nodetypes with different operating systems and hardware must be tested. To speed up the certification process we need to be able to quickly produce pre-defined images of different gLite nodetypes to use with Xen. We produce such images on a weekly basis in order to reflect the latest updates. The tools libfsimage and OS Farm were developed at CERN to achieve the aforementioned goals.

2.2 Usage of Virtual Machines on the Grid

The EGEE/WLCG infrastructure lets users send jobs for execution to more than 250 sites. Using the information system a user can determine the operating system provided by a site. However as more and more users from different scientific communities join the grid it gets difficult for sites to fullfill all their requirements in terms of operating systems and installed software. One way to deal with this problem is to provide each job on the grid a virtual machine with a dedicated OS setup. With thousands of users on the grid transferring images to sites becomes an issue. The content based image transfer described in Sect. 5 shows how to overcome this problem.

3 Libfsimage

Libfsimage is a library of Linux file system generation routines implemented in Python. Its primary goals are the simultaneous generation of 32-bit and 64-bit file systems for different Linux distributions in an isolated environment and reuse of the common setup and configuration code between the distributions. At present, the supported Linux flavors are Debian[6], Ubuntu[7], CentOS[8], Scientific Linux CERN and Fedora[9]. The package installation and dependency resolution for the first two are done with Debian package management tools. For the RPM[10] based distributions this is achieved with Yum[11].

Libfsimage uses pre-built environments that reflect the hardware architecture as well as the type and version of the package manager used in the relevant version of the distribution being generated. When possible these pre-built environments are shared between the different distributions. Prior to the generation the appropriate environment is deployed and the library uses the chroot system call to switch to a new root directory. The initial package installations in the generated file system are done from there. Once the latter contains the basic libraries, a package manager and configuration tools, further package installations and image configuration are performed from inside the new file system, again by leveraging the chroot capability.

Libfsimage can be used both from the command line and as a python library. In the latter case it manages a Workspace object that keeps track of the deployed environments and their status in order to speed up the file system generation by reusing them.

A consequence of the extensive use of the chroot system call by Libfsimage, and Debian and RPM based package managers is the root privileges indispensability. This is a major drawback in the scenario when Libfsimage is used for simultaneous creation of images for different third parties that need to install custom packages inside the generated file system. A harmful pre- or post-install scriptlet that is run during the installation of a package could escape the chroot jail and run arbitrary code with root privileges, thus compromising the host and interfering with the generation of other parties' images. To address this issue a SELinux policy module is being developed for narrowing the standard root capabilities to the required minimum and confining the concurrently running generation processes in dedicated SELinux domains, so that misuse of the CAP_CHROOT privilege can not lead to a system compromise.

4 OS Farm

OS Farm creates VM images and Virtual Appliances (VA) [12] that can satisfy different needs. It provides a web interface through which users can choose between a range of Linux distributions and yum repositories, with their corresponding yum packages. Images are generated using libfsimage, which provides a rich selection of Linux distributions, which in OS Farm are called *classes*.

The main interface to an OS Farm service is a web interface, which is shown in Fig. 1. It provides several ways for the user to request a VM image. The most

Fig. 1. OS Farm user interface

```
<image>
    <name>MySlc4Image</name>
    <class>SLC4</class>
    <architecture>i386</architecture>
    <package>emacs</package>
    <package>unzip</package>
    <group>Base</group>
    <group>Core</group>
</image>
```

Fig. 2. OS Farm VM image specification

basic method is a *Simple request*, which allows the user to select image class and architecture.

Advanced request extends *Simple request* and gives the possibility of adding yum packages to the image. A dropdown menu allows the user to choose between a list of predefined yum repositories. Using Asynchronous JavaScript and XML [13], a further list is returned which allows the user to select the corresponding yum packages.

OS Farm also supports image requests by XML descriptions. An image description can be uploaded as an XML file. An example image description is shown in Fig. 2.

4.1 Layered Generation and Caching

The generation of an image is divided into three layers or stages, *core*, *base* and *image*. *Core* is a small functional image with a minimal set of software required to run the image on a Virtual Machine Monitor or in order to satisfy higher level software dependencies. *Base* is a layer on top of *core*, which provides some additional software needed in order to satisfy requirements for VAs. An *image* is also a layer on top of *core* and provides user defined software in addition to the *core* software. A subclass of *image* is *virtual appliance*, which is an image with an extra set of rules aimed to allow the image to satisfy requirements of

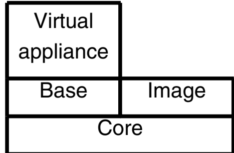


Fig. 3. Layers of a VM image

the deployment scenario. Fig. 3 shows that *core* can be shared between images, while *base* can be shared between VAs.

In order to accelerate the generation of images, *core* and *base* are always cached the first time they are generated. The layers are cached in Logical Volume Manager (LVM) [14] volumes. This allows higher layers to continue instantaneously, using copy-on-write, with the snapshot feature of LVM. *Images* are also cached and tagged such that if an image request matches that of a cached image, the image is returned immediately.

The tag of a cached image is the checksum of its configuration parameters, such as architecture and yum packages. Whenever an image is requested, a checksum is generated from the requested configuration and looked up in the cache. If an image with the exact same configuration parameters already exists in the cache, the image is returned immediately. A timeout value can be set on a cache entry in order to limit the validity of it. If an entry has timed out, a request for that entry results in a regeneration of it.

The cache also serves as a browsable repository for images. Instead of requesting images by parameters, images can be browsed and downloaded directly from the cache.

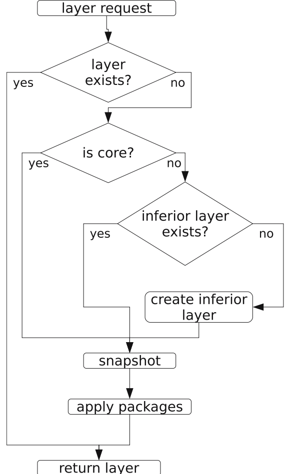


Fig. 4. Flowchart of the recursive image request and generation process

Fig. 4 shows the recursive image generation process. A request for an image is effectively a request for a layer, which in turn requires the presence of its inferior layer *core*, which, if it does not exist, triggers its creation.

The speedup gained from using these techniques varies between image configurations. We have measured the execution times of three simple example scenarios:

- the cache is empty, i.e. the image is generated from scratch: 254 seconds
- *core* is in the cache: 72 seconds
- *image* is in the cache: instantaneous

The results show that a significant speedup can be gained when an image or one or more of its layers are cached.

5 Content-Based Transfer

Content Based Transfer (CBT) is a technique to efficiently transfer VM image data from a source host to a target host. It takes advantage of knowing the structure of the image data to extract common data which need not be transmitted.

Most filesystems are aligned on a fixed boundary. For example, in the Ext2 filesystem, a file is aligned in blocks of size $1024*2^n$, where $0 \leq n \leq 2^{32}$ [15]. This means that two identical files on two different ext2 volumes, will be contained in a set of identical blocks on both volumes, even if they are fragmented. Moreover, if block sizes are different on the two volumes, as long as the largest block size of the two volumes is a multiple of the smallest block size, all files in both volumes will be aligned on the smallest block boundary. In our experience, block sizes of 1024 and 4096 are most frequent.

[16] examines the commonality between filesystem volumes. In our experiments we have analysed two different scenarios:

- two computer centre batch systems,
- two major versions of Scientific Linux CERN (SLC) VM images

Our experiments have shown that commonality ranges from moderate to significant. The commonality between some example volumes is shown in Tab. 1.

Before transmitting a volume, A, across the network, a comparison can be done between an existing volume at the destination, B, and A. If any blocks in A already exist in B, then those blocks need not be transmitted. Moreover, any

Table 1. Commonality between example volumes

Scenario	Commonality
Batch system to batch system	84 %
SLC4 to SLC3	48 %
SLC3 to SLC4	22 %

blocks in A which exist in any of the volumes at the destination, need not be transmitted.

Content Adressable Storage [17] exploits commonality in order to reduce storage load. In [18] this is exploited in order to reduce the load on the network and storage. We present an implementation which exploits commonality in order to reduce network load and speed up network transfer to close to the theoretical maximum.

5.1 Implementation

Our implementation of Content-Based Transfer uses Java, as a good compromise between efficiency and convenience. Most notably, it exploits Java's hash digest algorithms and thread-safe hash tables.

Identical blocks are identified with checksums, which are calculated using the available hash digest algorithms in Java. In spite of discovered collisions in the MD5 algorithm[1] [19], for the results in this paper, we have used it because it is the fastest available algorithm in the Java library. The choice of hashing algorithm, however, is open to the user.

The implementation is split into several threads that pass checksums and blocks among each other in a pipelined fashion. For example, one thread scans the source image and generates a checksum for the current block. Immediately, before continuing to the next block, the checksum is passed to the next thread. Concurrently, another thread, at the destination node, receives a checksum and looks it up in its hash table. If the block is not already at the destination, then it is requested from the source, in a similar pipelined fashion. The key to achieving good efficiency in this implementation is to allow the blocks that are already at the destination be written simultaneously and at the same pace as they are read from the source (which should be the disk read speed), and the other blocks be transmitted at the pace which the network allows.

5.2 Hypothetical Maximal Observed Bandwidth

The information needed to be transmitted from the source to the target consists of a *differential*, s_D, and an *identical*, s_I, part. In the differential part, all data is different between the source and the target, so all source data must be transmitted, and thus the transfer speed for the differential part is bound by the network transmission bandwidth, v_n, given that the disk speed is higher than the network bandwidth. In the identical part, data is identical between the source and the target, so data must only be identified and only identification information needs to be transmitted, and thus the transfer speed for the identical part is bound by the disk read speed, v_d.

Given an I/O bound only program (CPU speed is infinite, CPU time is 0), the hypothetical best transfer time is

[1] MD5 is not recommended for security application, since a collision can actively be created. This concern is not that relevant for our application because the user is not expected to actively create blocks that will collide with his or her own blocks.

$$t = \frac{s_I}{v_d} + \frac{s_D}{v_n} \quad , \tag{1}$$

iff

$$v_d > v_n \quad . \tag{2}$$

The total transfer time of an image, using this technique, is different from a non-optimized technique. With the optimized transfer time, we calculate an *observed bandwidth*. The hypothetical best observed bandwidth is, given (2),

$$v = \frac{s}{\frac{s_I}{v_d} + \frac{s_D}{v_n}} \quad . \tag{3}$$

In general,

$$v = \frac{1}{\frac{\Delta_I}{v_d} + \frac{\Delta_D}{v_n}} \quad , \text{where } \Delta_j = \frac{s_j}{s} \in [0,1] \quad . \tag{4}$$

The intent of the hypothetical best observed bandwidth is to determine the performance of a hypothetical optimal CBT implementation to use as a benchmark for our CBT implementation. “Observed bandwidth” serves as a measure of performance that is independent of image sizes and indicates the speedup given by the CBT technique compared to a non-optimized technique.

On our test system we measured, using the Unix command “dd,” a disk read speed of 35.6 MB/s. The test systems were also equipped with a 100 Mb/s network interface card, and the theoretical max bandwidth of a 100 Mb/s line is 11.9 MB/s. Using these v_d and v_n bandwidths, we calculated the *hypothetical* observed bandwidths which are given in Fig. 5.

5.3 Experimental Analysis

Running our implementation of CBT on our test systems, we measured the observed bandwidths which are given in Fig. 5. The results show that our CBT

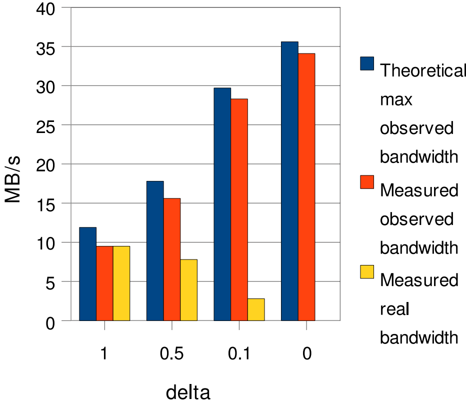


Fig. 5. CBT bandwidth

implementation achieves an observed bandwidth which closely follows the hypothetical maximum bandwidth. Also, the *real bandwidth*, which indicates the network load produced by the actual amount of data transmitted, is reduced, meaning a reduced load on the network.

It is worth noting that the observed bandwidth at the two extremes of Δ_D, at 0 and 1, are close to v_d and v_n, respectively. In our example, $v_d > v_n$. If $v_n > v_d$, as would be, for us, the case with a gigabit network, the disk read speed is the limitation, and CBT does not give any speedup. CBT in this case still has the advantage of reducing the load on the network.

6 Related Work

RPath[20] is a service that offers VAs, and provides a service similar to OS Farm. RPath's "recipe" approach to constructing a VA is a powerful method and gives great opportunity for reuse of packages between VAs. The approach encourages a VA developer, who develops recipes, and a VA user, who downloads the VAs, as two separate roles. The user can choose between a set of predefined VAs, but does not actively change the VA. In OS Farm, the user would normally also be the author of an image, since it is a trivial exercise.

If VM images are to be deployed on a large scale, they need to be adapted to their deployment context. Libfsimage allows paremeterized configuration of the images, but a future goal is to allow for contextualization[21].

Rsync[22] is an application that uses commonality in order to speed up the transmission of data. It uses SSH for authentication, which adds some overhead. The observed bandwidth given by Rsync, as calculated from the time reported by Rsync itself, which is lower than the total execution time including authentication, is not as high as CBT. For example, for $\Delta_D = 0.1$, Rsync gives a 30 MB/s observed bandwidth, and $\Delta_D = 0.5$ gives a 13 MB/s observed bandwidth. Another advantage CBT has is that it takes a set of images as a source for commonality, as opposed to Rsync, which uses only one target file.

7 Conclusion

We have presented tools and techniques for managing images, which help to overcome some of the problems that present themselves when managing an infrastructure of VMs. Libfsimage provides the different flavors and architectures that are needed in our use case and has a basis which allows extension for further flavors of Linux. It also lends itself as a library for external application, as in the example of OS Farm.

OS Farm uses Libfsimage and provides a graphical user interface for generation of VM images. It also provides a repository for VM images, which also serves to cache and optimize image generation through sharing layers of images.

CBT exploits commonality between images to optimize the transfer of images across the network. It achieves an observed bandwidth close to the theoretical maximal observed bandwidth. It can also help to avoid network congestion when transferring large images.

Acknowledgements

This work makes use of results produced by the Enabling Grids for E-sciencE project, a project co-funded by the European Commission (under contract number INFSO-RI-031688) through the Sixth Framework Programme. EGEE brings together 91 partners in 32 countries to provide a seamless Grid infrastructure available to the European research community 24 hours a day. Full information is available at http://www.eu-egee.org.

References

1. About EGEE, http://public.eu-egee.org/intro/
2. gLite, http://glite.web.cern.ch/glite/
3. Worldwide LHC Computing Grid, http://lcg.web.cern.ch/LCG/
4. European Organization for Nuclear Research, http://public.web.cern.ch/public/
5. Scientific Linux CERN, http://linux.web.cern.ch/linux/
6. Debian home page, http://www.debian.org/
7. Ubuntu home page, http://www.ubuntu.com/
8. CentOS home page, http://www.centos.org/
9. Fedora home page, http://fedoraproject.org/
10. RPM home page, http://www.rpm.org/
11. Yum home page, http://linux.duke.edu/projects/yum/
12. Saputzakis, C., et al.: Virtual Appliances for Deploying and Maintaining Software. In: Proceedings of LISA 2003 (2003)
13. Wikipedia article on AJAX, http://en.wikipedia.org/wiki/AJAX
14. LVM2 Resource Page, http://sources.redhat.com/lvm2/
15. Bovet, D.P., Cesati, M.: Understanding the Linux Kernel, pp. 574–607. O'Reilly, Sebastopol (2003)
16. Tolia, N., et al.: Using Content Addressing to Transfer Virtual Machine State (2002), http://www.intel-research.net/Publications/Pittsburgh/050520030704_127.pdf
17. Tolia, N., et al.: Opportunistic Use of Content Addressable Storage for Distributed File Systems. In: Proceedings of USENIX 2003 (2003)
18. Nath, P., et al.: Design Tradeoffs in Applying Content Addressable Storage to Enterprise-scale Systems Based on Virtual Machines. In: Proceedings of USENIX 2006 (2006)
19. Mikle, O.: Practical Attacks on Digital Signatures Using MD5 Message Digest, Cryptology ePrint Archive: Report 2004/356 (2004)
20. rPath home page, http://www.rpath.com/
21. Bradshaw, R., et al.: A Scalable Approach To Deploying And Managing Appliances. In: TeraGrid 2007, Madison, WI (June 2007)
22. Rsync home page, http://samba.anu.edu.au/rsync/

Dynamic on Demand Virtual Clusters in Grid

Mario Leandro Bertogna[1], Eduardo Grosclaude[1], Marcelo Naiouf[2], Armando De Giusti[2], and Emilio Luque[3]

[1] Department of Computer Science
Universidad Nacional del Comahue
C.P. 8300. Buenos Aires 1400. Argentina
{mlbertog,oso}@uncoma.edu.ar
[2] Informatic Research Institute LIDI
Universidad Nacional de La Plata, Argentina
{mnaiouf,degiusti}@lidi.info.unlp.edu.ar
[3] Computer Architecture and Operating System Department
Universidad Autónoma de Barcelona, Spain
Emilio.Luque@uab.es

Abstract. In Grid environments, many different resources are intended to work in a coordinated manner, each resource having its own features and complexity. As the number of resources grows, simplifying automation and management is among the most important issues to address. This paper's contribution lies on the extension and implementation of a grid metascheduler that dynamically discovers, creates and manages on-demand virtual clusters. The first module selects the clusters using graph heuristics. The algorithm then tries to find a solution by searching a set of clusters, mapped to the graph, that achieve the best performance for a given task. The second module, one per-grid node, monitors and manages physical and virtual machines. When a new task arrives, these modules modify virtual machine's configuration or use live migration to dynamically adapt resource distribution at the clusters, obtaining maximum utilization. Metascheduler components and local administrator modules work together to make decisions at run time to balance and optimize system throughput. This implementation results in performance improvement of 20% on the total computing time, with machines and clusters processing 100% of their working time. These results allow us to conclude that this solution is feasible to be implemented on Grid environments, where automation and self-management are key to attain effective resource usage.

1 Introduction

Using geographically distributed clusters in a coordinated manner has a major impact in execution time for parallel applications. Grid computing is a natural environment to deal with this usage, as Grids provide resource sharing through open standards and tight security, making possible to solve problems faster and efficiently.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 13–22, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The Grid metascheduler is an active component in distributed systems coordination and management. This component facilitates the user's tasks to access resources across different administrative domains. It can take decisions based on information of the whole system. Owners of physical resource become services providers, and the metascheduler orchestrates them according to negotiated policies and service level agreements. Virtual machines provide a way to make this task easier. These virtual machines can be independently instantiated and configured beforehand with sandbox-like environments[4, 9]. They also allow dynamically tuning of parameters like memory, number of CPUs assigned to each virtual machine, etc. Today's virtual machine technologies performance at CPU intensive tasks is comparable to that of native applications [5].

This paper presents a framework extension to a Grid metascheduler. The extension consist of two modules; the first one that dynamically discovers free machines determined by user requirements. The other one creates virtual clusters to efficiently satisfy submition of parallel jobs. The first module selects the clusters using graph heuristics. Free resources are mapped to a graph, machines as nodes and network links as edges. The algorithm then tries to find a solution by searching a set of clusters that achieve the best performance for a given task. The second module, of which one instance resides in every grid node, monitors and manages physical machines. When a new task arrives, these modules modify virtual machines configuration to dynamically adapt resource distribution at the clusters, thus obtaining maximum utilization. Metascheduler components and local administrator modules work together to make decisions at run time to balance and optimize system throughput.

In the second section of this paper we present the sequence of use and the architecture of the solution, focusing on the metascheduler; the third section describes the model, heuristics and algorithms developed; the fourth section presents experimental results; and finally, related works on this subject and conclusions are shown.

2 Metascheduler

From the design viewpoint, the architecture[1] of this solution is conceptually divided into three layers or tiers. In the first one, named *access tier*, the clients accessing the system are defined. The second, *management tier*, considers access control and creation of resources. Finally, the third, *resource tier*, deals with the implementation of physical and virtual resources.

This paper focus on the management tier cover by the metascheduler. The implementation begins with the study of several proposals, for this work, CSF (Community Scheduler Framework)[2] was chosen. CSF is an open-source implementation of a number of Grid services, which together functionally perform as a Grid metascheduler, and can be used as a development toolkit.

To satisfy on demand virtual clusters there are two extension modules within CSF. The first one is the *Resource Manager Adapter Service* called *GramVM*. This module is in charge of looking for free machines in the group of clusters.

For management and instantiation of virtual machines within local domains, a new module called *Hypervisor proxy* was implemented. Unlike the original CSF proposal, several *Hypervisor proxy* instances can be working towards one *GramVM* instance at the same time, in a coordinated manner. In the original implementation, CSF could work with different local schedulers, just one at every time.

Also different from the original CSF is that local schedulers, with similar duties as *Hypervisor Proxy,* have to be setup previously to CSF execution, and once execution in a cluster is started, the assignment of machines can not be modified until end of execution. For dynamically instantiated virtual machines, we do not know how many machines each cluster will have until the requirement arrives. *GramVM* should try to find the resources that best fit the task requirements, leading to a great number of alternatives. Besides, virtual machines can modify their usage of physical resources during execution, either by live migration[6], or by dynamically varying memory and CPU allocation. All of these features essentially reconfigure the pool of available free resources. They can be used to obtain better cluster performance and they are negotiated between *Hypervisor Proxy* and *GramVM* at run time.

3 Model

Finding a group of machines with specific characteristics, which is able to efficiently share a given workload, in a short time, is not a trivial problem. To approach this task, we settled for two criteria which were given higher priority: how fast the problem was solved, and how good the outcome was when compared to the optimal solution.

To be able to solve the problem in an analytical way, groups of clusters and free machines in the Grid environment are mapped to a graph. Machines are viewed as nodes and network links as edges. Nodes and edges have weights corresponding to machine features and bandwidth. More bandwidth-capable edges receive less weight. Node's weights are based upon cost functions such as per-time billing, computing power, etc.

The strategy is divided into two stages. The first one consists in selecting the groups of clusters. At this stage, a heuristic is used to find an optimal set of machines, taking into account communication overhead and machines computing power. Once a group of clusters is obtained, the second stage starts. For each cluster, an analysis must be done to evaluate how many physical machines will be incorporated. If the number of machines involved is greater than needed then efficiency will decrease, as some machines will stall waiting to send data to another cluster. We seek to keep efficiency over a certain threshold, given beforehand. Our analysis extends the work done in [3]. This work modifies the MPI library to span a number of clusters; a certain, unique, type of task is assumed. In our paper, a virtual environment is proposed where applications can run unmodified over a combination of clusters. Different types of tasks can be supported, as expected from a Grid environment.

Our model was evaluated over master-worker parallel applications. Clusters are dedicated and serve a previously determined number of tasks. All tasks within a same requirement from a user perform the same computation, and send or receive the same amount of data, but the number of tasks can vary across requirements. This schema is common in graphic and simulation parallel applications. It is assumed that each time a cluster is added to the grid environment, virtual images are characterized and a performance benchmark is done. The network links are monitored regularly to sense bandwidth changes.

3.1 Machine Selection Algorithm

To select a set of feasible clusters to be incorporated into the solution, an iterative improvement algorithm is used. This algorithm, known as *Hill-Climbing*, is mainly a loop that continually moves in the direction of increasing value; in this case, the direction maximizing computing power. The algorithm does not maintain a search tree; the node data structure needs to record just the last state reached. This simple policy has some drawbacks: *local maxima* (peak values that are lower than the highest peak value in the state space); *plateaux* (a region in the state space where the evaluation function is essentially flat) and *ridges* (a ridge may have steeply sloping sides, so that the search reaches the top of the ridge with ease, but the top may slope only very gently toward a peak).

The problem we are trying to solve has particular features, as certain networked geographical regions or provider domains are better provisioned than others. This geographical connectivity pattern is mapped onto the graph edges. When taking this feature into account, there is no need to do random restarts as in the original algorithm. If the graph is partitioned into better-connected geographical zones, or islands, the chances to find the global maximum grow, and the time to find it decreases. This modification is called *Hill-Climbing with k-restarts*, where k is the number of partitions on the graph. Each partition will be a starting point.

To partition the graph in geographical zones, a different kind of algorithm is used, namely *Minimum Spanning Trees (MST)*. A minimum spanning tree includes all nodes in the graph, such that the sum of their weighted edges is lesser or equal to that of any other spanning tree over the graph. The chosen algorithm is Kruskal's variant because of the approach taken to build the MST. This algorithm starts by sorting the edges by weight; then all nodes are agglomerated, starting from as many partitions as nodes. Traversing over the edges, the solution is checked at every iteration for cycles. If a cycle appears, the edge that was most recently introduced is discarded.

To enhance *Hill-Climbing* performance, we need to know how many restarts there will be. The number of restarts will be the number of suitable partitions in the graph. Once the number of partitions is set as a threshold, *Kruskal*'s algorithm starts adding edges until the threshold is reached. When *Kruskal* algorithm stops, the remaining graph is partitioned into maximally well-connected trees, as the first step taken was to sort the edges by weight. For each partition the *Hill-Climbing* algorithm is then applied, obtaining a global maximum.

The number of partitions depends on the number of nodes and the surface where *Hill-Climbing* algorithm will run. As a good practice the graph was partitioned until each segment had at least one complete cluster. In most cases that number of partition was three, this number assures to find the global maximum in each test.

The algorithmic complexity for MST is $O(E\ log(V))$. The *Hill-Climbing* algorithm using adjacency lists is $O(E\ log(V))$ where E are Edges and V are Vertices of the graph. Performance can be improved if the graph nodes are Grid nodes instead of machines, as the number of vertices in the graph decreases.

3.2 Cluster Usage Optimization

A parallel application in a multicluster environment is either limited by performance of machines in the cluster (compute-bound) or by network throughput (communication bound). The maximum performance (*maxperf*) is reached by an application on a particular cluster when it is compute-bound. If the application is communication bound, machines will sit idle waiting for network input/output. For a worker task running on a processor, the computation time (T_{Cpt}) is defined as the ratio between the task number of operations ($Oper$) and the processor performance ($Perf$): $T_{Cpt}=Oper/Perf$. The communication time (T_{Comm}) is the ratio between the volume of data communication ($Comm$) (worker task data from and to the master) and the network throughput (T_{Put}): $T_{Comm}=N*Comm/T_{Put}$. The *maxperf* is the performance that can be obtained when $T_{Cpt} >= T_{Comm}$.

Once the set of clusters is computed by the heuristics, the second stage starts. Here we analytically determine how many machines will be used in each cluster, so as to avoid *maxperf* dropping under a previously fixed threshold. This calculation is based on how many task's data the network is able to transfer by time unit, and how many tasks per time unit the cluster is able to process.

If the cluster processes more tasks than the network would transfer, then the application becomes communication bound; if the network is able to transfer more task's data than the cluster processes, then the application becomes computation bound. Hence, if the application is communication bound and the number of machines diminishes until a balance is reached, the cluster resources are not fully used; but the machines processing the tasks will be used at a maximal efficiency.

In a Grid environment, multiple possibilities for cluster assignment exist. If we regard the application as being started from different clusters (i.e. we choose different *master-clusters*), the resulting outcome from our analysis will be different. With the heuristic algorithm, this search work is minimized, and the best solution (reaching the highest computing power with a combination of clusters) is probably achieved. To make this possible, an analytic search has to be done for each graph partition. This will limit the number of machines for each cluster so that performance can be held over threshold for every machine. Not only communication and computing power for the local cluster has to be evaluated, but for the *master-cluster* as well. If the *master-cluster*'s bandwidth is smaller than the aggregated *worker-clusters* bandwidth, then machines from the

worker-clusters will be idle even though their own network links are enough to exploit their full computing capacity for a given task. Here, a fractional of the *master-cluster* bandwidth will be determined, based on the computing power of each cluster; and the analytic evaluation of computation/communication will be carried on upon this value.

This solution focuses on maintaining machine performance, but cluster usage is also a matter of importance. If a cluster can always satisfy a task with low computing requirements but high data communication, and the network link does always limit the computing power to a few machines, then this is not a good solution. An approach to this problem is to limit even more the usage of computing power, so as to free bandwidth. When tasks with less communication requirements arrive, they can be submitted to idle machines, thus improving cluster usage; on the downside, if such tasks never arrive, resources will be wasted. Our proposal in this paper is to build a cluster over virtual machines to release bandwidth on demand; when a new task with smaller communication requirements arrives, machines already executing on the clusters are migrated without interrupting the execution. When two or more virtual machines are executing on a physical processor, the virtualization software scheduler will assign computing resources fairly, so this will result in less computing power per machine and less data per task will be sent. The time for completion of both the new task and the executing task are known, so the metascheduler module can calculate the time gain for machine migration and will submit the new task onto the cluster. If the new task has smaller communication requirements than the migrated one, not only the physical nodes that were supporting virtual machines running on them, but also the idle physical nodes in the cluster, could be assigned to the new task, improving the whole cluster usage.

4 Experimental Results

The experimental evaluation is divided into two phases. The first one shows improvement gains by using the heuristic of machine selection. This strategy is compared to classical grid algorithms[7, 8] as a set of independent tasks arrives. From the system's point view, a common strategy is to assign them according to the load of resources in order to achieve high system throughput. Three algorithms were selected: a) Minimum Execution Time (MET): assigns each task to the resource with the best expected execution time for that task, no matter whether this resource is available or not at the present time b) Minimum Completion Time (MCT): assigns each task, in an arbitrary order, to the resource with the minimum expected completion time for that task and c) Opportunistic Load Balancing (OLB): assigns each task, in arbitrary order, to the next machine that is expected to be available, regardless of the task's expected execution time on that machine . The intuition behind OLB is to keep all machines as busy as possible. One advantage of OLB is its simplicity, but because OLB does not consider expected task execution times, the mappings it finds can result in very poor makespans. Classical grid algorithm like Min-min and Max-Min were not

selected because these begins with the set of all tasks and in this case this data is unknown.

The problem of grid scheduling can be investigated by taking experimental or simulation approach. The advantages of performing actual experiment by scheduling real applications on real resources are that it is easier and straightforward to compare the efficacy of multiple algorithms. However, in experimental study of scheduling algorithms it is seldom feasible to perform a sufficient number of experiments to obtain meaningful results. Furthermore, it is difficult to explore a variety of resource configurations. Typically a grid environment is highly dynamic; variations in resource availability make it difficult to obtains repeatable results. As a result of all these difficulties with the experimental approach, simulation is the most viable approach to effectively investigate grid scheduling algorithms. The simulation approach is configurable, repeatable, and generally fast, and is the approach we take in this work. Our simulation takes as parameters a description of the existing clusters, their network links, machines therein (specifying memory and processor type) and finally the tasks with their execution time for each type of processors.

For model validation a real test in [3] was considered, where four clusters with three, five and eight machines were used. The first two clusters were physically lying in South America, having less bandwidth and machines with smaller computing power. The third, more powerful one, was in Spain. If we enforce the same *master-cluster* as in the real test, i.e. the application is submitted from each of the clusters in South America, the final results in computing time and network throughput returned by the simulator are the same. However, if the simulator is used along with the machine selection algorithm, the Spanish cluster is selected and the total execution time decreases nearly by 50%. The explanation being, if the application is submitted from a South American cluster, then there will be idle machines in the Spanish cluster; while if the application is submitted from Spain, then the three clusters will have better usage.

4.1 Machine Selection Experiences

Tests were done to verify the impact of partition and *master-cluster* selection carried over by the *Hill-Climbing* algorithm. These tests compare how the heuristic algorithm proposed in this paper performs against classical grid algorithms. The tests were done simulating clusters composed of eight machines each one. Two arrival statistical distribution were chosen; the first one, a uniform distribution simulating low rate of arrivals; the other one, an exponential distribution simulating a incremental rate of tasks arrivals. Nineteen request were made, each one with three hundred process to be distributed into the clusters. In all cases MCT performs better than MET and OLB. For greater clarity, figure comparisons were done between Hill-Climbing and MCT algorithms.

The results of simulation executions can be seen in figure 1. In the firts graph, *Hill-Climbing* uses all clusters machines, selecting the best master cluster and MCT only selects the minimum completion time cluster. Choosing to run the application in all machines heuristic algorithm performs 70% better than MCT,

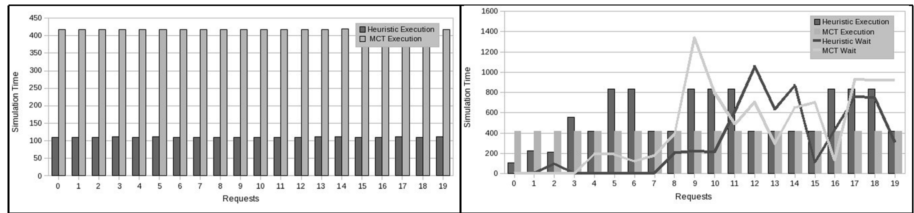


Fig. 1. Time execution comparison between Heuristic and MCT algorithm with uniform task arrival and exponential task arrival

this one is the best case. If each parallel task sends more data while processing, then the application becomes communication bounded ($T_{Cpt} <= T_{Comm}$) limiting the number of machines each cluster could use to compute, decreasing the algorithm performance. Once the algorithm proved to work with low rate of arrival the next step was to test it incrementing this rate. This can be seen in the second graph. Until the second request the execution time is below MCT. For some request the execution time are above MCT. This happens because tasks are distributed over different clusters. If tasks waiting times are observed (shown as continuous lines), MCT has longer waiting periods, compensating for faster executions. The average total time tends to be the same. After several simulations we can conclude that in the worst case *Hill-Climbing* heuristic tends to perform the same as MCT. In the average scenario (low rates of arrival with peaks at regular intervals), the *Hill-Climbing* heuristic algorithm performs with a 20% of improvement.

4.2 Cluster Adaptation Experiences

In cluster adaptation test, different task types were submitted requiring differents volumes of I/O. Task mixes were done with exponential rate of arrivals. Some requests overlap in time but waiting times were not too high. Because of this they are not shown in the graphs.

After several simulation executions it was noticed that some clusters can not be 100% used because tasks with high network I/O requirement took little percentage of clusters machines while saturating Internet connection. In the case where tasks with low network requirements arrive, this can be a waste of computing resources (as blocked machines could be used with smaller bandwidth). To solve this problem a migration procedure was proposed, if a virtual machine consuming network bandwidth was migrated to other physical machine already working, taking advantage of virtual machine live migration. The fair scheduling algorithm used in hypervisors will slow down computing power freeing network bandwidth, and these free bandwidth will be taken by the new process. This procedure is advantageous if the time waste in the migrated task is less than the time gain processing the new task.

In figure 2 we depict *Hill-Climbing* heuristic execution time and MCT execution time with a similar behavior as the previous figure with a 25% of improvement, and

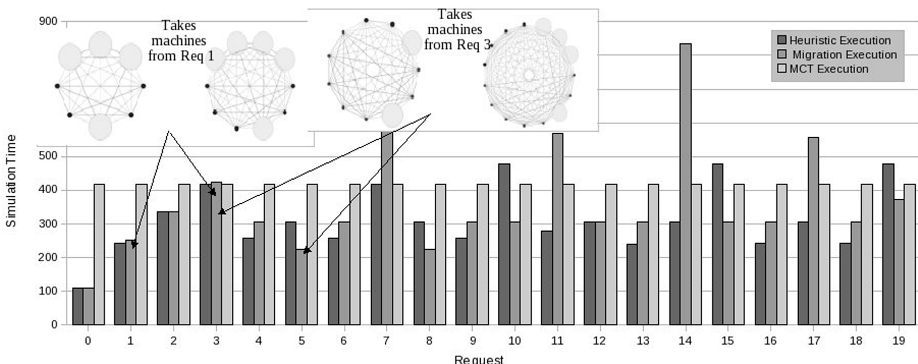


Fig. 2. Migration process sample

an example of the migration procedure. When Req3 arrives, the algorithm investigates a) where machines can be migrated and b) what the impact should be on the running application, were these machines actually migrated, i.e. whether the gain upon migration would pay for migration overhead. These conclusions can be drawn only having the application's run time characterized beforehand. In the boxes of figure 2 two examples of this procedure can be seen. In the upper left part of the box, a graph is shown with the machines originally assigned to the task (in this case Req3 and Req5) and then the machines after the virtual machine migration procedure. In the first case eight machines were assigned and then two machines from blocked clusters were added. A small increment can be noticed in the time in the execution of Req1 because of the slow down in the computing power after the migration. The same can be seen in Req5. The time gain is more obvious there. In the rest of the simulation, several cases exist with a longer execution time. This ocurrs because of different configuration of machine assignment after the migration process. On average, the performance improvement is of 5% over the originally proposed algorithm but this depends on tasks balance and arrival rate.

5 Conclusion

This paper has presented a metascheduler framework extension to generate high performance laboratories with virtual machines, local resource managers and management heuristic to obtain effective usage of clusters and machines. The framework takes into account not only the time taken by a task to complete but also network consumption, with the purpose of taking advantage of the bigger number of machines available in a grid environment. Geographical partitions through *Kruskal* graph algorithm also address the problem of scalability, decreasing the complexities in the search of the optimal solution. *Hill-climbing* with *k-restarts* does not ensure reaching an optimal solution; but in the tests done the best solution was achieved in nearly every case.

Tests were done by sweeping a range of arrival rates, cluster computing power, number of tasks, number of process per tasks and task computing and I/O requirements. Selection algorithms have been implemented to find groups of clusters that satisfy certain requirements in a small search space, making an effort to return a solution in a fast and optimal way. These implementations increase computing power by nearly 20%. Dynamic algorithms have been implemented to adapt cluster configuration at run time with migration of virtual machines. This implementation results in performance improvement of 10% on the total computing time, with machines processing 100% of their working time.

These results allow us to conclude that this solution is feasible to be implemented on Grid environments, where automation and self-management are key to attain effective resource usage. Where clusters serve fixed applications, multi-cluster analysis could guide balance tuning between computation and communication, determining whether it is more effective to either increment/decrement the bandwidth in use, or increment/decrement engaged computing power.

References

[1] Grosclaude, E., Luro, F.L., Bertogna, M.L.: Grid Virtual Laboratory Architecture. In: VHPC Euro-Par 2007 (2007)

[2] Open source metascheduling for Virtual Organizations with the Community Scheduler Framework (2004), `http://www.cs.virginia.edu/~grimshaw/CS851-2004/Platform/CSF_architecture.pdf`

[3] Argollo, E., Gaudiani, A., Rexachs, D., Luque, E.: Tuning Application in a Multi-cluster Environment. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 78–88. Springer, Heidelberg (2006)

[4] Foster, I., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B., Zhang, X.: Virtual Clusters for Grid Communities. In: CCGrid 2006 (2006)

[5] Barham, P.T., Dragovic, B., Fraser, K., Hand, S., Harris, T.L., Neugebauer, R.: Xen and the Art of Virtualization. In: SOSP 2003, pp. 164–177 (2003)

[6] Clark, C., Fraser, K., Hand, S., Hansen, J.G.: Live Migration of Virtual Machines. In: Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (2005)

[7] Dong, F., Akl, S.G.: Scheduling Algorithms for Grid Computing: State of the Art and Open Problems,Technical Report No. 2006-504, Queen's University, Canada, 55 pages (2006)

[8] Zhu, Y.: A survey on grid scheduling systems, Technical Report, Computer Science Department of Hong Kong University of Science and Technology (2003)

[9] Ruth, P., McGachey, P., Xu, D.: VioCluster: Virtualization for Dynamic Computational Domains. In: Proceedings of the IEEE International Conference on Cluster Computing, Cluster 2005 (2005)

Dynamic Provisioning of Virtual Clusters for Grid Computing*

Manuel Rodríguez[1], Daniel Tapiador[2], Javier Fontán[1], Eduardo Huedo[1], Rubén S. Montero[1], and Ignacio M. Llorente[1]

[1] Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid, Spain
{manuelro,jfontan}@fdi.ucm.es, {ehuedo,rubensm,llorente}@dacya.ucm.es
[2] European Space agency ESA
Villanueva de la Cañada, Spain
Daniel.Tapiador@sciops.esa.int

Abstract. Virtual machines can greatly simplify grid computing by providing an isolated, well-known environment, while increasing security. Also, they can be used as the base technology to dynamically modify the computing elements of a grid, so providing an adaptive environment. In this paper we present a Grid architecture that allows to dynamically adapt the underlying hardware infrastructure to changing Virtual Organization (VO) demands. The backend of the system is able to provide on-demand virtual worker nodes to existing clusters and integrate them in any Globus-based Grid. In this way, we establish a basis to deploy self-adaptive Grids, which can support different VOs in shared physical infrastructures and dynamically adapt its software configuration. Experimental results on a prototyped testbed show less than a 10% overall performance loss including the hypervisor overhead.

1 Introduction

Recently, interest in virtual machines is quickly growing, as hardware support provided by new generation microprocessors significantly reduces the overhead of virtualization [1]. Typically, hypervisors like Xen [2] or VMWare [3] take advantage of these new hardware features to improve their performace. Computational Grids can greatly benefit from virtualization. A Grid is a highly heterogeneous system both in terms of the hardware and software configuration of its components. This fact reduces the number of potential resources to run a given

* This research was supported by Consejería de Educación de la Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE), through BIOGRIDNET Research Program S-0505/TIC/000101, by Ministerio de Educación y Ciencia, and through the research grant TIN2006-02806, and by the European Union through the research project RESERVOIR Contract Number 215605.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 23–32, 2009.
© Springer-Verlag Berlin Heidelberg 2009

application, which usually requires specific versions of different software components (e.g. operating system, libraries or post-processing utilities). Moreover, the installation, configuration and maintanace of these different components significantly increases the operational costs of the Grid infrastructure. Finally, those organizations that contribute resources to a Grid usually want to limit the interaction of Grid applications with their own *internal* workload.

Among other solutions, Virtual Machines (VM) can solve the aforementioned problems. From the user perspective, VMs ensure the correct execution of the application by encapsulating software configurations in a "well-known" environment. On the system administrator side, VMs are an efficient technology to isolate and partition the system. Thus, allowing them to set the amount of resources devoted to Grid jobs. Also, the operational cost of the infrastructure is reduced as specific appliances to run an application class can be prepared, configured and deployed.

The integration of virtual machines in Grid environments has been previously explored by several works. For example, the In-VIGO project [4] establishes a basic layer of virtual Grid resources upon which any grid middleware can be deployed. The Virtual Workspace Service [5], exposes the functionality needed to manage workspaces –abstraction of execution environments implemented through VMs. Also, a straightforward deployment of virtual machines to execute scientific codes in a Grid has been analyzed in [6] (see Section 2 for an additional description of other related works).

In this work we propose a novel architecture for the dynamic provisioning of computational services on a Grid infrastructure. The system leverages virtualization technologies to provide flexible support for different Virtual Organizations (VO). Usually, the resources of a Grid site support different VOs (e.g. Bioinformatics or High Energy Physics). The proposed system is able to balance the amount of resources allocated to each VO in terms of their dynamic requests.

On the other hand, different VOs need different software, or even different versions of the same software. Traditionally, the cost of the installation, configuration and maintainace of VO-specific worker nodes have limited the flexibility of the infrastructure. In our case, the system will deploy on-the-fly VO-specific worker nodes to execute their applications.

To achieve these two goals, we propose a multi-layer architecture to provide virtual worker nodes to clusters inside a Grid. The infrastructure, given a set of virtual machine images, is capable of deciding the number and the kind of worker nodes to be created on each cluster. This way, the computing elements of a grid can be adapted to fit the changing software requirements and computational demands. Section 3, provides a detailed description of the system.

The paper also analyzes, in Section 4, a prototype implementation of the architecture. In particular, we discuss the overhead and interaction of all the components of a classical middleware stack, namely: local resource management systems (Sun Grid Engine in our case), Grid resource services (Globus GRAM), information services (Globus MDS4) and meta-schedulers (GridWay). Finally, in Section 5, we discuss our experience and explain our future work.

2 Related Work

The idea of a virtual cluster which dynamically adapts its size to the workload is not new. Jeffrey Chase et al., from Duke University, describe in [7] a cluster management software called COD (Cluster On Demand), which dynamically allocates servers from a common pool to multiple virtual clusters. Although the goal is similar, the approach is completely different. COD worker nodes employ NFS to mount different software configurations. In our case, the solution is based on VM, and the system is studied from a Grid perspective.

Ananth I. Sundararaj et al., from Northwestern University, have also worked on dynamic cluster configuration [8] with a different approach. They have developed a network tool that connects virtual machines, making the connectivity problem identical to that faced by the user when connecting any new machine to his own network. This allows to migrate VMs across different domains while preserving the same IP. Our aim is not to develop a totally virtualized architecture, but to include the advantages of virtualization in existing infrastructures in a non-intrusive way.

OSG (Open Science Group) defines Edge Services, that mediate access between a site and the external world. They can handle common grid operations like job submissions or data movement. I.Foster et al. [9], employ VMs to bring them up dynamically only as they are needed. Although this work is also based on virtualization technology and the Virtual Workspace Service (VWS) the approach is different. In this paper, our goal is not to create a new service (Edge Service) but to improve and adapt existing ones, supporting different VOs in a shared physical infrastructures.

Finally, Amazon Elastic Computing Cloud [10] provides a remote VM execution environment. It allows to execute one or more “Amazon Machine Images” on their systems, providing a simple web service interface to manage them. Users are billed for the computing time, memory and bandwidth consumed. This service greatly complements our development, offering the possibility of potentially unlimited computing resources. It would be possible to employ a grid-enabled Amazon Machine Image, and create as many instances as needed, getting on-demand resources in case the physical hardware cannot satisfy a peak demand.

3 Description of the Architecture

In this section we detail the architecture of the system and its basic behavior. In its design, we have focused in avoiding dedicated systems. The virtual worker nodes provided by the underlying physical infrastructure can register in existing clusters queues (computing services). So the flexibility of the Grid infrastructure can be boosted up without requiring dedicated hardware, or modifying neither existing applications nor software configurations. The final goals of the proposed architecture are:

- Dynamically adapt a shared infrastructure to support different VOs, by balancing the physical resources allocated to each VO.

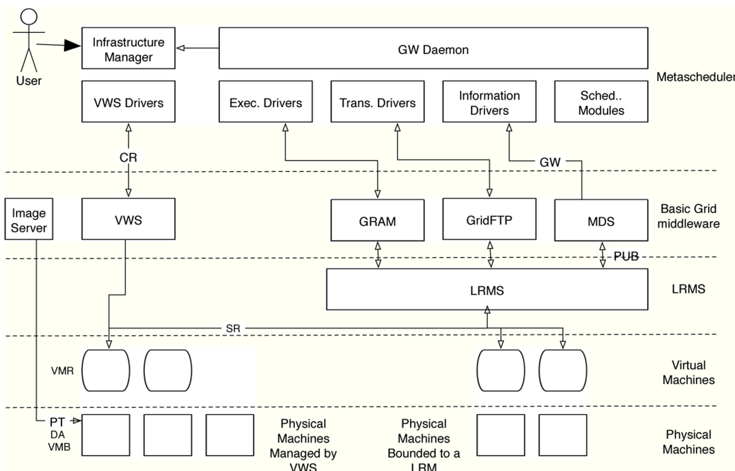


Fig. 1. Architecture Overview. Arrows represent an intertaction between components, the time of this interaction is labeled in the figure.

- Reduce the operational cost of the Grid infrastructure, by providing a simple way to provide on-demand software configurations to VO users.
- Minimize the *Gridification* time, by executing VO applications in a well-known pre-defined environment.

The base of the architecture (depicted in Figure 1) is the Virtual Machine Layer. This layer is responsible for the creation of virtual worker nodes to be registered in a cluster. Typically, the functionality of this layer will be provided by a hypervisor. In the present work we will consider Xen 3.

The worker nodes (either physical or virtual) are managed by a Local Resource Management System (LRMS layer in Figure 1). In our case we will use a Sun Grid Engine (SGE [11]) instance. The SGE cluster is configured with a different queue for each VO, which also provides a VO appliance. So, when a new worker node is to be deployed the corresponding appliance is used. This way, several software configurations can coexist on a single cluster and a job can be executed in the correct one by just specifying the queue name. We would like to note that having a queue for each VO is a usual configration for Grid resources (for example in the EGEE [12] infrastructure).

The previous resources are exposed as Grid services by the components of the third layer, Basic Grid Middleware. The prototype version considered in this paper uses the services provided by the Globus Toolkit 4, namely: information (MDS4), execution (GRAM4) and file transfer services (GridFTP); along with a virtualization interface (Virtual Workspace Service, VWS). The ability of VWS to deploy several copies of a single image along a physical cluster, with a pre-set pool of host names and IPs, keeps integration with LRMS layer simple.

At the top layer (Management) we usually find meta-schedulers that manage, control and monitor the execution of Grid applications. We have chosen the

GridWay [13] meta-scheduler. As, it provides some features which are required by the proposed architecture:

– Scheduling capabilities: GridWay employs a dynamic scheduling system. It can detect when a new machine has been added or removed from a cluster, and redistribute the work load.
– Fault detection & recovery capabilities. Transparently to the end user, GridWay is able to detect and recover from any of the Grid elements failure, outage or saturation conditions [14].

3.1 Infrastructure Manager

Finally, the Infrastructure Manager module completes the system architecture. This component is responsible for adapting the Grid computational services to the dynamic Grid computing demands. The Infrastructure Manager decides when to add new worker nodes (and their type) to a given computing element (cluster queue). So, allowing Grid administrators to adapt their services according to a pre-defined set of policies (e.g. the cluster should be shared in a 2 to 5 ratio between the fusion and bioinformatics VOs.)

The following sequence of actions (see Figure 1) describes the actions that takes place when the Information Manager decides to add a new worker node to the computing element:

1. The Infrastructure Manager request a new VM to the VWS (using a pre-defined appliance for the VO). The VWS determines the best node to run the virtual machine, based on the resources requested (e.g memory). The arrow labeled "CR" in Figure 1 represents the time since the Infrastructure Manager sends a deploying petition until it receives a confirmation.
2. If the VM image (appliance) is not local to the host system, it accesses the image server via a suitable protocol (e.g. GridFTP) and obtains a copy. We will refer to the time employed on the image transmission as "PT".
3. Once the image has been transferred, the physical node's DHCP server configuration file is altered in order to establish VM's IP and hostname. The time to assign a hostname and IP will be referred as "DA".
4. When these operations conclude, the VM is booted. "VMB" denotes the time since the hypervisor (Xen) receives the execution instruction until it starts booting the VM. Also, "VMR" is the time need to actually boot the system. Note that while VMB is constant, VMR highly depends on the virtual machine configuration and services.
5. When the VM has been deployed and is running, it registers on LRMS frontend (arrow "SR" in Figure 1) as an execution host.
6. After a given time, the Grid information system (MDS) detects the new node and publishes it. This step is labeled "PUB" in Figure 1.
7. Finally, the meta-scheduler (GridWay) will refresh the information of the available Grid resources, and detect the new worker node (label "GW"). Then, according to the scheduling policies, it will allocate jobs on this new resource by interfacing with the Grid execution service (GRAM).

4 Experimental Results

4.1 Testbed Description

The behavior of the previous deployment strategy will be analyzed on a testbed based on Globus Toolkit 4.0.3 and GridWay 5.2.1. The testbed consists of three resources: two SGE clusters, and a dedicated system hosting the management services (meta-scheduler and infrastructure). The main characteristics of these machines are described in Table 1.

Table 1. Characteristics of the testbeds resources

Host	OS	CPU	Memory	Services
UCM frontend	Debian Etch	P4 HT 3.2GHz	1 GB	GT4.0.5, SGE NIS, NFS, VWS
UCM WN (2)	Debian Etch	2 x P4 HT 3.2GHz	256MB	DHCP, Xen 3
ESA frontend	Fedora Core 6	Xeon 1.70GHz	768MB	GT4.0.4, SGE NIS, NFS, VWS
ESA WN 1	Fedora Core 6	2 x Xeon 2.20GHz	2GB	DHCP, Xen 3
ESA WN 2	Fedora Core 6	Xeon 1.70GHz	768MB	DHCP, Xen 3
ESA WN 3	Fedora Core 6	Xeon 2.20GHz	2GB	DHCP, Xen 3
Manager Server	Debian Etch	Pentium M 1.4GHz	768MB	GT4.0.3, GridWay 5.2.1

4.2 Functional Analysis

The aim of the experiments presented in this section is to obtain a clear understanding of the interaction of all the components that form the architecture. Also, we will study the overhead induced by each component, so we can evaluate the the cost of the benefits that virtualization adds to the Grid in terms of flexibility, lower operational costs and enhanced security.

Deployment Overhead. Let us start by measuring the deployment time of a VM under several conditions. As different hardware configurations lead to different results, we have only employed one cluster (UCM, see Table 1) to carry out these tests. Table 2 presents the average results (over 25 runs) obtained while deploying one, two and three VMs on the same physical machine, respectively.

In Table 2, time precision is one second. Hence, the values obtained in the measurements of CR, DA, VMB, VMR and SR can be considered the same for the three experiments. We would like to note that the overhead induced by the SGE layer is negligible (the time to register a worker node in the cluster is less than a 1% of the total deployment time).

Although most of the overheads are constant regardless of the number of VMs deployed, the growth of propagation time (PT) makes this approach unfeasible. These results suggest to use a sequential deployment approach for worker nodes.

On a sequential deployment, VMs are deployed one at a time. Only one image is being transferred from the image server to the execution node, so I/O overhead is greatly reduced, obtaining nearly a constant time. Both approaches can be compared in figure 2. We would like to remark that when deploying more than 3 VMs simultaneously most of the times error situations occurred.

Table 2. Deployment time (in seconds) when starting one, two and three VMs simultaneously. CR: Command Received in Physical Node. PT: Image propagation time. DA: DHCP server alter. VMB: Xen start time. VMR: VM booting time. SR: SGE registering.

VMs Created	CR	PT	DA	VMB	VMR	SR	Total
1	2	96.8	4.58	3	10.67	1.5	118.58
2	2.54	279.58	4.82	6.44	11.73	1.88	308.02
3	1	472	3.75	5.8	11.83	1.5	495.88

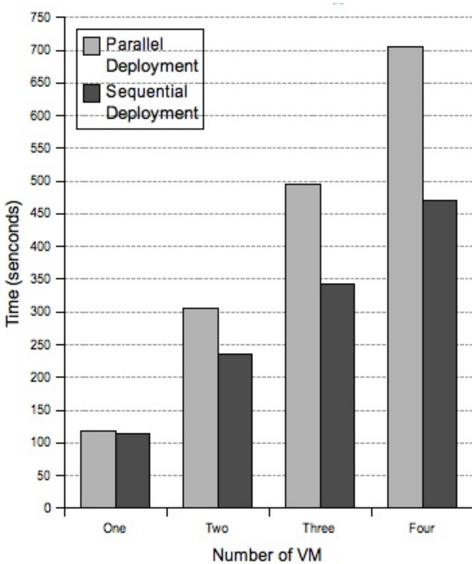


Fig. 2. Deployment overhead for simultaneous and sequential approaches

Shut Down Overhead. When shutting down a VM (Table 3), we have measured three relevant values. In this case the time is constant regardless of the number of VMs being shut down.

Overhead introduced by SGE when shutting down a VM can be minimized by reducing the polling time. Default value is 300 seconds, so it takes an average of 150 seconds to detect the new situation. Reducing polling time limits the overhead, although increments network usage. System administrator must tune this value according to the number of nodes in the cluster and average VM uptime.

Grid Integration. Figure 3 shows a global view of the interaction of all the system components. As has been discussed previously, the time to start a virtual worker node (time since the Infrastructure Manager requests a worker node, 114 sec., till it is registered in the LRMS, 2 sec.) is roughly 2 minutes.

Table 3. Times (in seconds) when a worker node is shut down

Number	Command Received	VM Destroyed	SGE	Total
1	0.78	6.22	145	152
2	0.88	6.46	158	165.33
3	0.67	7.33	151	159

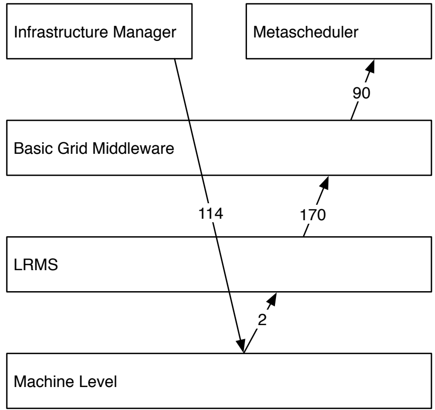


Fig. 3. Sequence of actions in a VM deployment and the associated overhead (in seconds)

The time to register the new slot in the Grid Information system (MDS4) is about 170 seconds. It is worth pointing out that MDS publishing time is greater than the time employed on deploying one VM plus SGE register time. Therefore, when sequentially deploying several VMs both times overlap, producing an additional time saving. The MDS and GridWay overhead can be limited by adjusting their refresh polling intervals.

When switching down, the same steps are accomplished. In this case, the time until the operation is accomplished at the machine layer is greatly reduced, from 114 to 7 seconds. However, time until LRMS detects the lack of the VM is incremented, from 2 to about 150 seconds. It is interesting to note that the meta-scheduler could assign jobs to the cluster during the worker node shutting down time. In this case the meta-scheduler should be able to re-schedule this job to another resource.

Virtualization Overhead. Virtualization technology imposes a performance penalty due to an additional layer between the physical hardware and the guest operating system. This penalty depends on the hardware, the virtualization technology and the kind of applications being run. Two good performance comparisons of VMware and Xen were conducted by the computer science departments at University of Cambridge [15]. and Clarkson University [16]. On these studies,

Xen performed extremely well in any kind of tasks, with a performance loss between 1 and 15%. VMware also achieves near-native performance for processor-intensive tasks, but experiences a significant slow-down (up to 88%) on I/O bound tasks.

VWS development team measured VWS performance in a real-world grid use case [17], a climate science application, achieving about a 5% performance loss. In a previous work [6], we have obtained a similar result (about 10% loss) when employing virtual machines in a Grid to execute a high throughput scientific application. This is an acceptable result, regarding the benefits in terms of modularity, portability and simplified application development.

5 Conclusions and Future Work

In this paper, we have presented a Grid architecture for the dynamic provisioning of computing elements. The benefits of this architecture is a flexible Grid able of supporting different VOs on a shared and configurable infrastructure, while reducing its operational costs.

The results obtained on the performance tests show that the proposed architecture and technologies represent a feasible solution. With a daily cluster reconfiguration, induced overhead is less than 1%. Added up with Xen performance lost, it remains under 10%. However, it provides attractive benefits like increased software robustness, easier cluster administration and enhanced security.

In the future, we will improve the decision making system on the resource manager, in order to optimize the deployment of virtual machines under different conditions and usage. We will also explore the possibilities that the proposed technology offers to outsourced grids, allowing dedicated service providers to supply resources on demand over the Internet.

References

1. `http://www.amd.com`
2. `http://www.xen.org`
3. `http://www.vmware.com`
4. Adabala, S., Chadha, V., Chawla, P., Figueiredo, R., Fortes, J., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L., Zhu, X.: From Virtualized Resources to Virtual Computing Grids: The In-VIGO system. Future Generation Computer Systems 21(6) (April 2005)
5. Keahey, K., Foster, I., Freeman, T., Zhang, X.: Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. Scientific Programming Journal 13(4), 265–276 (2005)
6. Rubio-Montero, A.J., Huedo, E., Montero, R.S., Llorente, I.M.: Management of virtual machines on globus grids using gridway. In: IEEE International Parallel and Distributed Processing Symposium 2007, vol. 21, pp. 1–7 (2007)

7. Chase, J.S., Irwin, D.E., Grit, L.E., Moore, J.D., Sprenkle, S.E.: Dynamic virtual clusters in a grid site manager. In: HPDC 2003: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, Washington, DC, USA, p. 90. IEEE Computer Society, Los Alamitos (2003)
8. Sundararaj, A.I., Dinda, P.A.: Towards virtual networks for virtual machine grid computing. In: VM 2004: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium, Berkeley, CA, USA, p. 14. USENIX Association (2004)
9. Keahey, K., Freeman, T., Rana, A., Norman, M., Wrthwein, F., Gardner, R.: Edge services framework for osg. OSG Document, 167-v1 (June 2005)
10. http://www.amazon.com/ec2
11. http://gridengine.sunsource.net
12. http://www.eu-egee.org
13. http://www.gridway.org
14. Huedo, E., Montero, R.S., Llorente, I.M.: Evaluating the Reliability of Computational Grids from the End User's Point of View. Journal of Systems Architecture 52(12), 727–736 (2006)
15. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Alex Ho, R.N., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: Symposium on Operating Systems Principles, pp. 164–177 (October 2003)
16. Clark, B., Deshane, T., Dow, E., Evanchik, S., Herne, M.F.J., Matthews, J.N.: Xen and the Art of Repeated Search. In: USENIX Annual Technical Conference, p. 47 (2004)
17. Foster, I., Freeman, T., Keahy, K., Scheftner, D., Sotomayor, B., Zhang, X.: Virtual clusters for grid communities. In: CCGRID 2006: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2006), Washington, DC, USA, pp. 513–520. IEEE Computer Society, Los Alamitos (2006)

Dynamic Resources Management of Virtual Appliances on a Computational Cluster

Alexander A. Moskovsky[1], Artem Y. Pervin[1], and Bruce J. Walker[2]

[1] Program System Institute, Russian Academy of Sciences, Pereslavl, s. Botik, Russia
[2] Hewlett-Packard Laboratories, Palo Alto 1501 Page Mill Rd, US
moskov@phys069b-2.chem.msu.ru, ArtemPervin@gmail.com,
bruce.walker@hp.com

Abstract. Virtual machine (VM) technology offers increased flexibility in resource provisioning. Load for applications typically varies over time, justifying the need for dynamic resource allocation/relinquish — exactly what VM technology allows. An approach for automated, dynamic resource management of applications running on a computational cluster has been devised. The job of the framework is to maintain a certain service level of application within tolerable limits. To do this the framework is able to dynamically vary resources available to the application. To facilitate performance optimization an application performance profile can be created using stress-testing tools. A software toolkit that allows running single and multiple VM applications has been created. Sample services (including both computing oriented and web oriented) have been tested and performance-resource dependences studied. We present an ongoing work on dynamic resource allocation, involving optimal control and optimization methods.

Keywords: virtual environment management, service level agreements, virtual machines cluster, virtual appliances.

1 Introduction

The research goal is to develop methods and tools to deal with multiple applications sharing a computational cluster in a virtualized environment. With regular clusters, the number of nodes occupied by a particular application serves as the main resource consumption metric. In addition to this the virtual machine (VM) technology offers several new capabilities bare iron environments couldn't feasibly offer. In particular using `Xen` one can:

1. Suspend a VM to disk and resume it later, allowing one to utilize the resources for a higher priority service.
2. Pause a VM, leaving it in memory, to allow running some other application/VM.
3. Live migration of a VM from one host to another for any one of a variety of reasons (e.g. consolidate resources to allow something new to run).
4. Start a VM with some number of virtual CPUs and then add or remove CPUs on the fly based on need and the priority of other VM.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 33–42, 2009.
© Springer-Verlag Berlin Heidelberg 2009

5. Start a VM with some fraction of each CPU it has allocated and then grow or shrink this fraction dynamically.
6. Start a VM with some amount of memory and then grow or shrink this footprint dynamically.
7. Start a VM with some I/O capacity and grow or shrink this capacity on the fly.

This drastic increase in flexibility allows not only "carve to order" (dedicate only as much resource to the task as needed) but also allows shared "over provisioning" instead of individual application "over provisioning". This should allow IT managers to provide considerably more services within the same infrastructure, particularly if one can automatically reallocate the resources based on their needs. As well, automation saves human time (and associated cost), human reaction can be too slow. At the same time, performance overhead of virtualization can be kept relatively low [1, 2].

Related works are numerous in the both industry and academia. `Amazon`s EC3`, `3Tera`s Applogic` are widely known industrial projects, which are capable of hosting on-line services in customized VMs. `Cluster-on-Demand` [3] provides a toolkit to create virtual clusters of VMs on top of physical machines. `Virtual Workspaces` [1] are utilizing VMs to isolate applications from hosts in a grid environment (starting grid jobs inside VM), leveraging and extending `Globus Toolkit`s resource management [4]. `SoftUDC` [5] is a utility computing platform, which virtualizes CPU, storage and network resources for applications running on a cluster. Resource sharing techniques in virtualized environments are sometimes borrowed from economics. `Shirako` [6] offers a mechanism were the application and the framework can negotiate and contract on resource leases. `Tycoon` [7] is an example of auction-based modeling applied to resource scheduling.

We believe that one step further is feasible: an automatic resource manager can be aware of how valuable given portions of the CPU, memory or some other resource are to a given application under the current load. With this information one can more accurately decide how to deploy incremental resources and when to request resources back from applications. In this work the software has been developed to conduct experiments with various resource allocation schemas. The following assumptions are considered in these schemas:

- Each application has a set of key parameters that define a quality of service for end-users, such as a response time for a web site application. These parameters can be measured at application run time.
- With the help of VM technology, various resources can be allocated dynamically with very fine granularity and thus they can be treated as continuous variables.

With these assumptions, the problem for managing the application's quality of service can be treated as an optimal control problem with continuous variables. This makes this work considerably different from other researches in QoS management [8-10]. In our work applications are considered as black-boxes and management framework can utilize powerful optimal control theory methods to implement efficient optimization mechanisms for resource allocation.

In order to make use of theoretical basis, we advocate the *service level* abstraction to enable automated decision-making about which resources are necessary for an

application to run. The runtime framework can receive sensor data about current application state and use application *performance model* to make decisions on how to maintain the service level for this application. In case of resource shortage, the framework can take away resources from less important (for end users) applications. Performance profiles can be either measured before runtime or generated on the fly.

The goal is to support a variety of types of single threaded and parallel services (virtual appliances) with desired performance characteristics under dynamically changing load conditions. This support implies optimal use of the resources so the greatest number of services can be launched on a given amount of hardware.

To accomplish this goal we first needed an infrastructure that allowed deployment, monitoring and resource re-allocation for virtual appliances. The next section of this paper describes the Virtual Services toolkit developed to provide such infrastructure.

Next the performance profile or models on the services are required. This information would not only describe the ideal resource mix (memory, number of virtual CPUs, CPU share value and I/O capacity) for a given load but would also describe the effect of adding or losing various resources. Section 4 outlines the concept of performance profiles. Finally we describe an arbiter that can take the performance profiles and the runtime information and re-allocate resources to optimize the effect of those resources.

2 Virtual Services Software

To accomplish the research goals a simple system was implemented that allows us to roll-out *virtual services* — parallel virtual appliances. Virtual appliances can be either online services (like web-site) or high-performance parallel applications. Other applications may include online gaming services, data processing, and transformation or retrieval applications.

Consider the components and interaction of those components in the fig. 1.

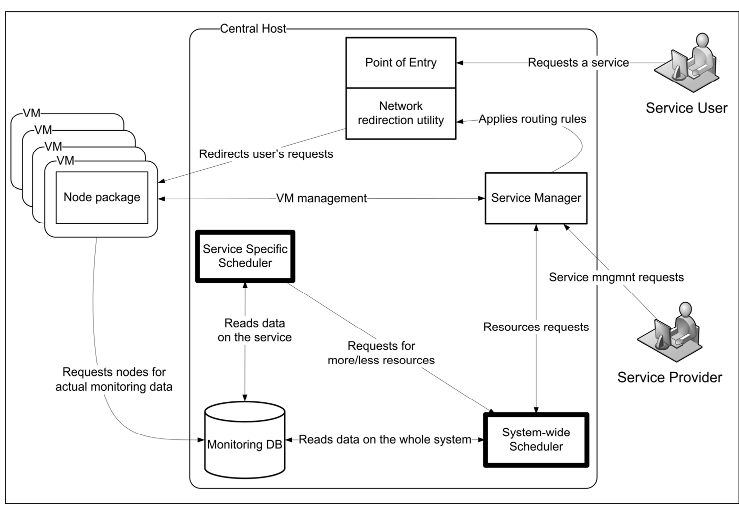


Fig. 1. Virtual service software components

A typical use-case of the system involves a Service Provider (an administrative user), who starts a service. The Service Users access it via the *point of entry* — an IP address accompanied with a port.

Capabilities of the software kit include, but not limited to:

- Instantiate/kill a service. Service instantiation involves launching one or more VMs with appropriate disk images and setting up of network traffic redirection rules and forming of virtual network for VMs of the service.
- Allocate/free the resources for the service. Resources request may originate from a human (Service Provider) or from the application-specific resource provisioning component which is known as *service specific scheduler.*
- Redirect network traffic. This is necessary to enable users' access to the virtualized application. It's also can be used for load-balancing.

The resource management flexibility is crucial for the research. At the same time, the ability to run generic algorithms on a system-wide layer is also important. Bearing this in mind, a two-layer resource scheduling mechanism had been designed to isolate concerns of system and service layers. On the lower layer a pluggable, service-specific scheduler makes resource distribution decision locally, considering monitoring information. On the top layer the *system-wide scheduler* takes into account agreements between Service Provider and a System Administrator, such as the Service Provider's priority, to provide globally optimal distribution of the resources between the services. In addition, the system-wide scheduler is able to query an application performance model, when it's necessary to evaluate variants of resource allocation. The schedulers are able to communicate with each other and access the monitoring data they need. The framework automatically maintains a certain resources quantity available to the application, in order to keep agreed service level. If a cluster node fails with a VM running on it, the framework will re-instantiate the VM on available free nodes using technique similar to high-availability clustering solutions. The Ganglia monitoring system is used to implement a heartbeat mechanism.

3 Service Examples

- **WebMapServer**
  This application [11] allows querying different information from geographical maps. In our tests the data on Itasca County, MN was used. It was derived, for the most part, from USGS (US Geological Survey) 1:24,000 quadrangles. The page displayed to the user includes a custom generated map in GIF format.

- **X-Com**
  The computational service is based on the `X-Com` utility. X-Com is a meta-computing framework, developed at Moscow State University [12]. The X-Com concept is similar to the `Condor` [13]. However, the implementation is simpler, light-weight, easy to install and use, yet applicable in wide variety of computational environments.

- **Virtual Cluster**
  The virtual cluster service starts the necessary number of VMs with the network connectivity support between each of them. Start of this service results in a set of the nodes that is called a *virtual cluster*. The network support is provided with the help of a bridging mechanism. It allows including a VM into the virtual cluster absolutely transparently for the user. The users may interact with the virtual cluster node without any extra effort as if it was a normal host.

A number of computational experiments were conducted on the virtual cluster. It was shown that in such environment one can successfully run LAM MPI applications that operate on several nodes concurrently . We have also run experiments with launching of parallel programs created with the help of rapid parallel application development tools such as `OpenTS` [14].

4 Performance Profile

An application performance profile represents the dependency between the amount of resources provided to the application, the user activity being generated on this application and the quality of service this application provides to the users. The amount of resource can be expressed in either absolute values (1GB of RAM) or in relative form (53% of CPU). The user activity is specific for different application classes. For example, for the web-site the user activity is a number of concurrent requests to a particular page of the web-site (that is, request rate). Finally, the quality of service can be measured as difference between desirable (or target) state of the service and its current state. The service level concept is discussed in details below.

This dependency can be expressed in tabular form. Tabular form describes some typical use cases of the application at different levels of user activity and the interpolation or extrapolation is used to obtain the values that are not provided in the table. The data for the tables can be collected with the help of stress-test utilities, such as `httperf`. We believe that this , given sufficient data collected in tables, may be appropriate for resources provisioning.

A number of experiments with various user activity and amount of resources dedicated to the service were conducted. Currently the following VM parameters are available to tune: amount of memory, number of virtual processors allocated by VM (VCPUs) and CPU cap, defining the maximum processor's time share the VM is allowed to occupy. The user load in each test run was chosen in a way to maximize the usage of resources provided to the service and at the same time keeping the network errors (such as request timeouts) at minimal levels. These user load values are referred as *inflexion points* — the highest load the service is capable of handling well.

The WebMapServer performance profiling demonstrated an insensitivity of this application to the amount of memory: only tiny variations of request rate were observed (fig. 2). Increasing VCPUs doesn't improve performance and eventually degraded performance (fig. 3). The application is apparently single threaded so we focused on the CPU cap parameter in the tests as the most significant VM parameter. The dependence of the maximum request rate from the CPU share is almost linear.

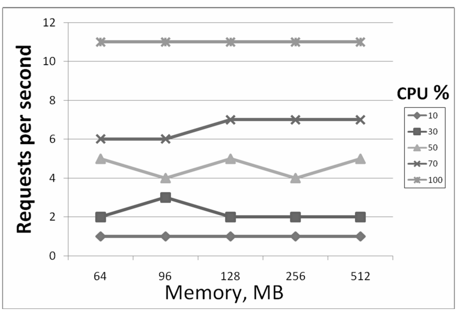


Fig. 2. Maximum requests/sec served by WebMapServer, at different amount of memory and Xen CPU caps

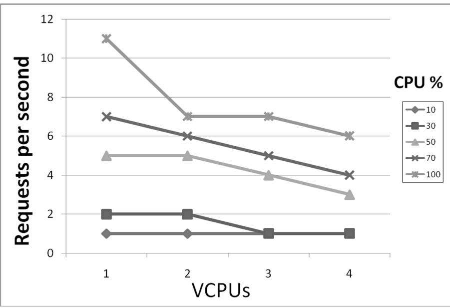


Fig. 3. Maximum requests/sec rate, served by WebMapServer, vs. VCPU of Xen at different CPU caps specified

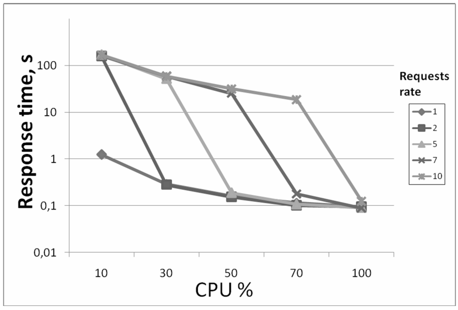


Fig. 4. Response time of WebMapServer vs. CPU cap specified in Xen configuration, under different load (requests per second)

One may see in fig. 4 that under 10 requests per second (the most right line), quality of service is quite sensitive to the CPU share provided to Xen: only above 70% share is response time tolerable. At the same time, 1 request per second can be served even with 10% CPU cap at reasonable response time (1 second).

The performance profiles can be used to evaluate variants of resource allocation without actually affecting the performance of applications running in the framework.

5 Service Level Agreements

Consider the situation when a web-site owner wishes to maintain service response time below a certain threshold (e.g. 1 second). If the web site is experiencing a spike of user activity, additional computing resources are required to keep the quality of service at a reasonable level. In this paper, this level is referred as a *target level*. It is natural to use an automated mechanism to maintain the target level since human intervention can be too slow and unreliable. One possible control schema is illustrated in the fig. 5 below. In order to re-use the optimization algorithms in the *Optimizer*, which acts on the system-wide layer, it is necessary to provide translation from application-specific parameters

(such as response time) to abstract service level value [15], which is done with the help of a *service level function*.

The idea of abstract service level function is the following. The function takes a *measured parameter* as an argument and results in a service level value that is in the 0% - 100% interval. Next, these values are passed to the Optimizer, who will find optimal resource allocation, maximizing service levels of the services by varying resources available to the services.

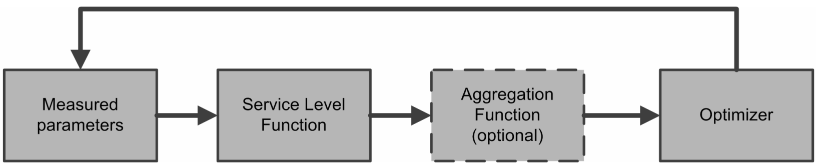


Fig. 5. Control Chain

The service level function is defined in a unique way for each service as the measured parameters and their target ranges differ from one service to another. Thus for the web-site service where the target service level is defined as the average response time below some reasonable value, the measured parameter should capture (at least) current response time. For the computational service with the deadline defined to finish the calculation, measured parameter describes an average calculation speed. In this case the optimal calculation speed will allow reaching the deadline without utilizing extra resources.

Since the service level functions are continuous, the powerful continuous optimization math algorithms can be applied in the Optimizer. In the general case the service level function is switch-shaped curve of the form:

$$s(x) = \frac{w * a * (b - x)}{\sqrt{1 + (a * (b - x))^2}} + w \tag{1}$$

where x — measured parameter (e.g. response time), s — service level, a, b and w — are parameters, allowing one to tune the curve's shape to fit application needs.

We suggest setting the value of 50% as the lowest tolerable level for any service in the framework. The Optimizer should try to keep all service levels above 50%. The value of 50 is selected because the changes on the service level function curve are the steepest in this area and hence optimization algorithms will be most sensitive to changes around this last tolerable value.

By tweaking the service level function parameters one is able to create soft or hard requirements to the function. Consider an example of service level function (1) with parameter a equal to 10, b — 1 and w — 0.5. This function is almost 100% below 1 second and rapidly vanishes to zero above 1.5 seconds (fig. 6). The curve parameters should be carefully chosen to fit application characteristics and its target service level. The service level concept can be easily generalized to include more parameters, e.g. percent of errors in addition to response time. In this case multidimensional function can be used.

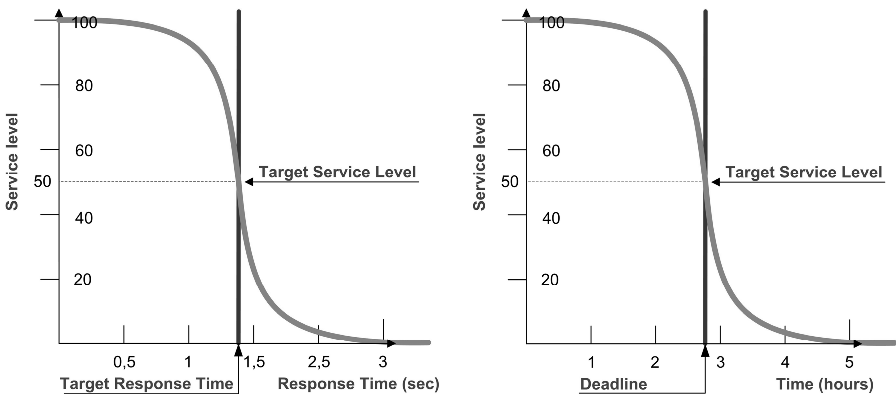


Fig. 6. Service Level Curve for the web-site service

Fig. 7. Service Level Curve for the computational service

It is clear that the service level approach is not only valid for web-site applications but can be extended to other application types. It can be applied for the job queues as well. As was mentioned before, the Service Provider may specify a deadline by which all jobs in the queue should be completed (see fig 7). Given job characteristics, it is possible to evaluate calculation speed and estimate completion time. If the deadline is met under current resources the service level is 50%. It will degrade until it becomes 0%, when results of the calculation are obsolete (say, weather forecast for tomorrow completed three months later).

In real life the framework should be able to deal with multiple applications simultaneously. The aggregation of n service levels into one can be achieved by weighted multiplication:

$$\sigma(x_1 \dots x_n) = \prod_{i=1}^{n} w_i S_i(x_i) \tag{2}$$

where the w_i are relative weights assigned to each application, S_i are application service levels and σ is the whole system service level. By maximizing σ the framework should enable tomorrows weather forecast service to be finished by afternoon at, probably, the expense of some inconvenience to web-site visitors in the morning. An array of optimization or optimal control methods can be applied in order to maximize σ - dynamic programming to name just a starting point.

6 Preliminary Implementation of Optimization Algorithm

So far, a naive one-dimensional constrained optimization is used to find the optimal resources required for a given application. Consider the example with the computational service. The algorithms below tries to figure out the minimum CPU share that still satisfies the target service level by using a binary search:

- First stage requires finding the boundaries of the range of CPU share.
- System starts with some guess on the resource necessary for the application. If the current service level is below the target level (case 1), the value of the guess CPU share will be higher than currently available to the service. Otherwise the algorithm will try to decrease the CPU share (case 2).
- This step will be repeated with increasing values of guess doubling the step until the current service level will not reach some value near the target level (higher than the target level in case 1, and lower in case 2).
- On the second stage, the algorithm uses the last 2 values obtained on the previous stage to perform a binary search in the range of these values to find an optimal CPU share distribution for the service. This step implies continuous increasing (or decreasing) CPU share available to the application and measuring of the new service level.

This algorithm is started periodically. In that way the service level of the application is kept at target value automatically. The existing implementation is simplistic, but still applicable to various services. Thus, for example, this algorithm was used to dynamically allocate resources for the computational X-Com service during its operation. The X-Com service was launched with some deadline to finish the job and the volume of resources a priori insufficient to complete the computation on time. However using this schema, the system increased the amount of resource dedicated to the service enough to finish the job close to the deadline. Our tests demonstrated only 0.5% deviation from the deadline. Thus a 40 minutes long computation was finished only 10 to 15 seconds late. The experimental setup was the following:

- Computational service CPU share initially was 40% of one CPU, at the end was 3.25 CPUs.
- Service was supplied with a set of uniform tasks.
- The optimization algorithm was launched once every 5 minutes.
- The hardware platform was 4-node cluster, each node has 2.0 GHz Opteron 175 (Dual-core) CPU and 2 GBytes of RAM, nodes are connected with Gigabit Ethernet.

7 Conclusions

We have created the platform where the central orchestrator and service specific scheduling agents can co-operatively create and remove instances of services and increase/decrease the resources for services. Simple automation based on service level concept has been demonstrated. More importantly, however, applications were tested on this platform in order to create performance profiles for more sophisticated automation software to leverage. This approach is not tightly knit to the Virtual Services and can be harnessed in other frameworks, (e.g. Virtual Workspaces, Cluster-on-Demand) after minor modifications.

One might argue that modifications on VM container CPU share, abrupt disconnections of VM instances may be detrimental to high-performance computing applications. It is true that most of existing MPI applications will suffer: they are written

with the assumption of uniform processor performance and could not sustain disconnection of even a single process. Nevertheless, more advanced parallel applications can live with this. `MapReduce` [15] is an example of a general purpose framework that can deal with the additional complexity of such a dynamic environment. With all the efforts spent by IT community to improve high-level parallel programming tools and techniques, one can expect such applications to become more widespread.

References

1. Keahey, K., Foster, I., Freeman, F., Zhang, X., Galron, D.: Virtual Workspaces in the Grid. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 421–431. Springer, Heidelberg (2005)
2. Youseff, L., Wolski, R., Gorda, B., Krintz, C.: Paravirtualization for HPC Systems. In: Workshop on Xen in HPC Cluster and Grid Computing Environments, Sorrento (2006)
3. Moore, J., Irwin, D., Grit, L., Sprenkle, S., Chase, J.: Managing Mixed-Use Cluster with Cluster-on-Demand, Technical Report (2002)
4. Sotomayor, B.: A Resource Management Model for VM Based Virtual Workspaces, Masters Paper, University of Chicago (2007)
5. Kallahalla, M., et al.: SoftUDC: A Software-Based Data Center for Utility Computing. Computer 37(11), 38–46 (2004)
6. Fu, Y., Chase, J., Chun, B., Schwab, S., Vahdat, A.: SHARP: An Architecture for Secure Resource Peering. In: 19th ACM Symposium on Operating Systems Principles (2003)
7. Lai, K., Rasmusson, L., Adar, E., Sorkin, S., Zhang, L., Huberman, B.: Tycoon: an implementation of a Distributed Market-Based Resource Allocation System. Technical Report, HP Labs, Palo Alto (2004)
8. Moroni, S., Joffre, A., Figueroa, N., Sahai, A., Chen, Y., Iyer, S.: A Game-theoretic framework for Optimal SLA/Contract Creation, HPL Tech. Report (2007)
9. Bennani, M., Menasce, D.: Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In: Second International Conference on Autonomic Computing
10. Menasce, D., Bennani, M.: Autonomic Virtualized Environment. In: International Conference on Autonomic and Autonomous Systems, p. 28
11. MapServer, `http://mapserver.gis.umn.edu/`
12. Voevodin, V.l., Filamofitskiy, M.: Supercomputer for a weekend, Open Systems. In: Otkrytie Sistemi, vol. 5, pp. 43–48 (2003) (in Russian)
13. Thain, D., Tannenbaum, T., Livny, M.: Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience 17(2-4), 323–356 (2005)
14. Abramov, S., Adamovich, A., Inyuhin, A., Moskovsky, A., Roganov, V., Schevchuk, Y., Schevchuk, E.: The T-system with an open architecture. In: Supercomputer Systems and Applications, Minsk, pp. 18–22 (2004)
15. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: 6th Symposium on Operating System Design and Implementation, pp. 137–150 (2004)

Complementarity between Virtualization and Single System Image Technologies*

Jérôme Gallard[1], Geoffroy Vallée[2], Adrien Lèbre[1], Christine Morin[1], Pascal Gallard[3], and Stephen L. Scott[2]

[1] INRIA Rennes - Bretagne Atlantique, PARIS project-team, Rennes, France
[2] Oak Ridge National Laboratory, Oak Ridge, USA
[3] KERLABS, Rennes, France
{jerome.gallard,adrien.lebre,christine.morin}@inria.fr
{valleegr,scottsl}@ornl.gov
pascal.gallard@kerlabs.com

Abstract. Nowadays, the use of clusters in research centers or industries is undeniable. Since few years, the usage of virtual machines (VM) offers more advanced resource management capabilities, using features such as virtual machine live migration. Because of the latest contributions in the domain, some may argue that single system image (SSI) technologies are now deprecated, without considering some complementarities between VMs and SSI technologies are possible.

After evaluating different configurations, we show that combining both approaches allows us to better address cluster challenges such as flexibility for the usage of available resources and simplicity of use. In other terms, the study shows that VMs add a level of management flexibility between the hardware and the application, whereas, SSIs give an abstraction of the distributed resources. The simultaneous usage of both technologies could improve the overall platform resources utilization, the cluster productivity and the efficiency of the running applications.

Keywords: cluster, virtualization, SSI, resource management.

1 Introduction

Clusters are today a standard computation platform for both research and production. Batch schedulers or single system image systems (SSI) are frequently used to manage clusters. In the first case, a head node is in charge of scheduling applications whereas in the second case, the SSI makes an abstraction of the cluster resources creating the illusion of an SMP machine. Several studies have focused on combining virtual machines (VMs) and batch schedulers in order

* The INRIA team carries out this research work in the framework of the XtreemOS project partially funded by the European Commission under contract #FP6-033576. ORNL's research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 43–52, 2009.
© Springer-Verlag Berlin Heidelberg 2009

to provide better resource control [1]. Features provided by virtualization technologies (such as isolation and suspend/resume) enable more advanced resources management capabilities. For instance, one VM can be suspended or migrated to another node. Thus, administrators are able to make maintenance operations without impacting the platform availability. On the other side, isolation mechanisms simply the management of security constraints.

This trend around virtualization seems to impact directly cluster management and more precisely SSI technology which enables, in some ways, similar capabilities. For instance, the openMosix SSI project [2] has recently closed. In that sense, we wonder whether virtualization technology will surpass the SSI systems or if these two models are complementary.

This paper addresses these questions and investigates in which extends the association of both virtualization and SSI technologies could improve the usage and management of distributed architectures as well as application execution (*e.g.,* administration, application debugging, and security).

To our best knowledge, virtualization and SSI approaches have been used only in the Peta-SSI project [3], using VMs in order to study the system scalability, "emulating" a large number of nodes. In other terms, only one capability (virtual machine stacking) provided by virtualization solutions has been studied. In this document, we analyze the potential benefits of all major capabilities provided by the usage of VMs in an SSI context. This study has been done in a theoretical way (no results of experiences are presented in this document).

The remainder of this paper is organized as follows: Section 2 clarifies the notion of virtualization and virtualization. Section 3 gives a brief background on SSI systems. Section 4 investigates the complementarity of virtualization and SSI. Section 5 reports lessons learnt. Section 6 concludes.

2 Introduction to Virtualization

Virtualization is an active research topic in operating systems (OS) since the 70's but regained popularity with the latest technologies which provide extra computational capabilities (new machines such as multi-core processors can compete with multiple individual servers that are few years old). A way to use this extra capabilities is to execute VMs on top of physical machines. From our point of view, the concept of VM includes five major features: (i) *isolation* (*i.e.,* degree of isolation between VMs, the bare hardware and applications running in different VMs), (ii) *server consolidation* (*i.e.,* capability of changing on demand resources allocated to a specific VM), (iii) *application portability* (*i.e.,* capability of executing an unmodified application), (iv) *virtual machine portability* (*i.e.,* capability of migrating virtual environments to different hardware architectures), (v) *suspend/restart* (*i.e.,* possibility to take a snapshot/resume of VMs).

Nowadays, two typical virtualization approaches are possible: one which implements the virtualization at the system-level (well-known Goldberg classification [4]) and the other at the process-level (*containers*). Part (a) of Table 1 summarizes functionalities supported by virtualization solutions.

Table 1. Selected Capabilities Enabled by Virtualization (a) or SSI (b)

	Virtualization (a)			SSI (b)	
	Container	Type-I Virt.	Type-II Virt.	Partial-SSI	SSI (full-SSI)
Isolation	-	+	+	-	-
Server conso.	+	+	+	-	+
App. Portability	-	+	+	-	-
VM Portability	-	-	+	-	-
Suspend/Restart	+	+	+	+	+

System-level Virtualization (Type-I, Type-II). This approach aims at virtualizing a full OS. For that, a virtual hardware is exposed to a full OS within a VM. The system running in a VM is named a *guest OS*. According to isolation properties associated with virtualization, the VM cannot execute privileged instructions at the processor level. To access the physical devices, drivers are hosted in a privileged OS, called *host OS*. Moreover, VMs run concurrently and their execution is scheduled by the *hypervisor*. The hypervisor is also in charge of forwarding all privileged instructions from VMs to the host OS.

Goldberg created a model for system-level virtualization, model based on two functions, ϕ and f. The function ϕ makes the correspondence between process running on the guest OS and the resources (exposed within the VM) whereas f makes the correspondence between resources allocated to a VM and the bare hardware. Based on those functions, Goldberg identified two different types of system-level virtualization: *type-I* (*e.g.,* Xen [5]) and *type-II* (*e.g.,* QEMU [6] and VMware [7]).

Process-level Virtualization (Container). It consists of running several processes concurrently on top of the same OS, each having its own view of available resources (*e.g.,* OpenVZ [8], *chroot* [9] and *containers* capabilities provided by recent kernels). The Goldberg classification is only focusing on the former level virtualization solutions and does not integrate such process-level virtualization solutions. In this paper, we only consider OpenVZ-like solutions. Recent kernel containers approach are not taken into account.

3 Introduction to Single System Image

An SSI is an OS that aims to abstract the distributed nature of the cluster in order to ease users, administrators and programmers tasks. For that, a SSI globally manages distributed resources. Because a transparent management of resources is difficult to implement at user-space (it is typically the responsibility of the OS), most of the SSIs are implemented at OS-level. Two kinds of SSI exist: (i) partial-SSI and (ii) SSI (or full-SSI).

Partial Single System Image. A *partial-SSI* only allows a global management of a subset of cluster resources (typically processes) and only from a central

location (*i.e.,* a "head node"). All processes are manipulable from this head node like if they were local processes; on other nodes, resources are still viewed as distributed. The abstraction of the distribution of resource is therefore "partial". Glunix [10], Bproc [11] or Cplan [12] are examples of partial-SSIs.

Full Single System Image. A *SSI* (or full-SSI) provides not only a global management of processes but also a global management of all other resources, and therefore gives to users the illusion to use an SMP machine. In such systems, there is no head node; each node is equal and has a global view of the distributed resources. Users are able to run SMP applications on the cluster without application modification or recompilation. For instance, SSIs implement a Distributed Shared Memory (DSM). This functionality enables the execution of OpenMP parallel applications based on the shared memory programming paradigm. Therefore, a SSI abstracts the complexity created by the resource distribution. Kerrighed [13] and OpenMosix [14] are examples of such SSIs.

The SSI technology has several interesting capabilities for cluster management, high performance, high availability, and, ease of use and programming. However, in this document, we focus only on the functionalities described in Section 2. Part (b) of Table 1 summarizes them.

4 Combining Virtualization and Single System Image

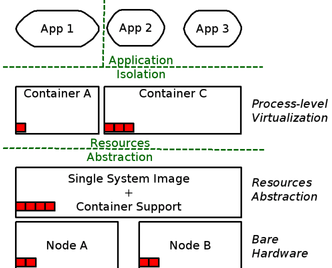
In this section, we present a systematic analysis of the combination of SSI and virtualization technologies. To realize this study, we selected three different target applications: (i) a web server such as Apache [15], (ii) an MPI-like application (based on message passing), and (iii) an OpenMP-like application (based on a shared memory). We think that these kinds of application are representative of a large part of business and scientific software. To achieve our main objective, we analyze the benefits of the five capabilities enabled by virtualization (cf. Section 2) with configurations exploiting SSIs.

4.1 Single System Image and Containers

Containers (*e.g.,* OpenVZ-like solutions) allow applications to be isolated from each other on the same node. Moreover, it is generally possible to assign an IP address, to allocate memory and CPU time to each container. Hence, a container could be migrated in most cases from one node to another.

Container Upon Single System Image. Figure 1 depicts the architecture of a typical system running containers upon an SSI. With this architecture, the SSI abstracts the distributed resources. Based on this "simplified" and "unified" view of the distributed system, global resources can be dynamically and transparently assigned to containers in order to fit at best applications needs. In other terms, containers could dispose of more resources that is available on one node.

Isolation: Applications are isolated from the bare hardware by containers that are running on top of the SSI. However, an application could hijack a container,



Little squares represent the amount of resources provided by the system. For instance, each node provides 2 resources and the SSI provides 4 resources (aggregation of resources provided by node A and node B).

Fig. 1. Containers Upon Single System Image

and in consequence, compromise the security of the whole system (there is one kernel for all containers). This property is not validated.

Server Consolidation: The SSI globally manages all resources, it is possible to change on demand the resources allocated to each container. These capabilities are very interesting for an Apache server administrator: according to the frequentation of a web site, it is possible to allocate more or less physical resources to the cluster (resizing the containers accordingly). This property is validated.

Application Portability (AP1): Thanks to the SSI, containers can span multiple nodes. Thus, an OpenMP application or an Apache server could take advantage of the SSI DSM (spanning nodes) whereas, an MPI application could take advantage of several containers each of them having their own IP address. Application portability is therefore guaranteed by such an architecture.

Virtual Machine Portability: Containers have not been designed to create a virtual hardware different from the hardware it is running on. Thus, the virtual machine portability is not validated.

Suspend/Restart: Containers can be suspended/restarted at any time by any other entity running with the correct privileges inside the system. Moreover, the SSI can suspend/restart any containers since a container is a set of standard resource from the SSI point of view. This property is validated.

Single System Image Upon Containers. Figure 2 presents the use of an SSI upon containers. The architecture is not realistic because no individual kernel can run in a container, only user-level applications can be hosted.

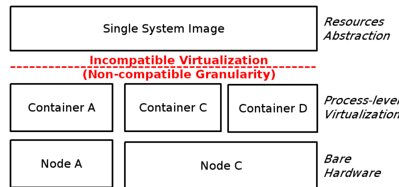


Fig. 2. Single System Image Upon Containers

4.2 Single System Image and Type-I Virtualization

Type-I virtualization solutions have an hypervisor running directly on top of the bare hardware and "hosting" the host OS and the VMs.

Type-I Virtualization Upon Single System Image. Figure 3 shows the architecture of a type-I virtualization solution running upon a SSI. This approach enables the implementation of a "global type-I hypervisor", including SSI features into the hypervisor. Such a global hypervisor can transparently and globally manage resources (creation of an SMP illusion) and typically the resource allocated to VMs is not restricted to the local resources.

Isolation (I2): The type-I hypervisor isolates applications from both the bare hardware and others VMs. For instance, if a hacker is able to become root on one VM, only the local VM is compromised: isolation is validated.

Server Consolidation (SC2): In case of a node addition, VMs can be moved to the new node; in case of node eviction, VMs can be transparently moved away. This propriety is validated.

Application Portability: Same as AP1, substituting containers by VMs.

Virtual Machine Portability: Currently no type-I virtualization solution provides emulation capabilities. Moreover, the SSI running on the side of the type-I hypervisor does not support by definition hardware architecture heterogeneity. It is therefore not possible to migrate VMs between nodes having different hardware architectures. VM portability cannot be achieved.

Suspend/Restart (SR2): Type-I hypervisor enables VM suspend/restart. However, if two applications are running in the same VM, it is not possible to suspend only one of them. Property of suspend/restart is not totally validated.

Single System Image Upon Type-I Virtualization. Figure 4 shows the architecture of an SSI upon the VMs of a type-I virtualization solution. In this

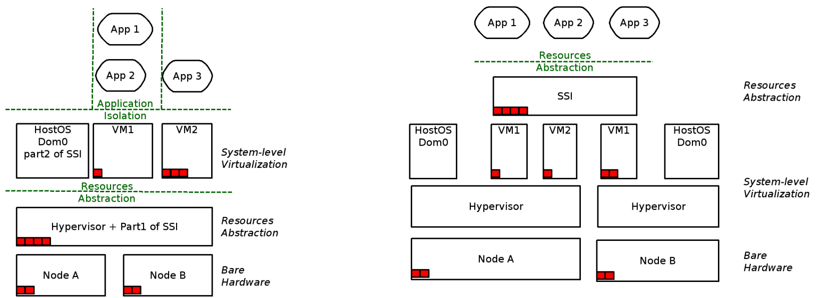


Fig. 3. Type-I Virtualization Upon SSI

Fig. 4. SSI Upon Type-I Virtualization

case, an hypervisor is deployed on all cluster nodes and the SSI is executed in different VMs; each VM being potentially hosted by different hypervisors.

Isolation (I3): The type-I virtualization isolates both the SSI and applications from the bare hardware. However, if the SSI is compromized the management of all resources and thus all running applications may be compromized. Isolation is therefore partially achieved.

Server Consolidation: The type-I hypervisor enables VMs migration and the SSI provides process migration between VMs. This property is validated.

Application Portability: Applications are actually running on top of the SSI, providing an SMP illusion. This enables the execution of MPI-like, OpenMP-like and Apache-like applications compiled for the native OS of the SSI. This property is validated.

Virtual Machine Portability: Today no type-I virtualization solution allows the emulation of an architecture at the VM level that is different from the bare hardware. Portability is, for the moment, not achieved.

Suspend/Restart (SR3): Both virtualization and SSI solutions provide suspend/restart mechanisms, respectively suspending/restarting VMs and processes. This property is validated.

4.3 Single System Image and Type-II Virtualization

Type-II virtualization solutions run VMs upon a host OS and generally provide live migration and suspend/resume capabilities.

Type-II Virtualization Upon Single System Image. Figure 5 shows the execution of VMs upon an SSI. The SSI globally manages all the distributed resources; the type-II virtualization hypervisor can therefore allocate distributed resources to VMs on demand in a transparent manner.

Isolation: Same as I2, substituting type-I hypervisor by type-II hypervisor.

Server Consolidation: Same as SC2, substituting type-I by type-II.

Application Portability: Same as AP1, substituting containers by VMs.

Virtual Machine Portability: It is possible to migrate a VM between nodes (according to VM resource needs) or SSIs compiled for other architectures. This property is validated.

Suspend/Restart: Same as SR2, substituting type-I by type-II.

Single System Image Upon Type-II Virtualization. Figure 6 shows the architecture of an SSI upon VMs. As for the type-I, each node runs VMs, and the SSI is deployed upon them.

Isolation: Same as I3, substituting type-I hypervisor by type-II hypervisor.

Server Consolidation: In case of node addition/removal, there are two cases: (i) an *automatic reconfiguration of the SSI* (*e.g.,* the SSI "knows" that nodes are

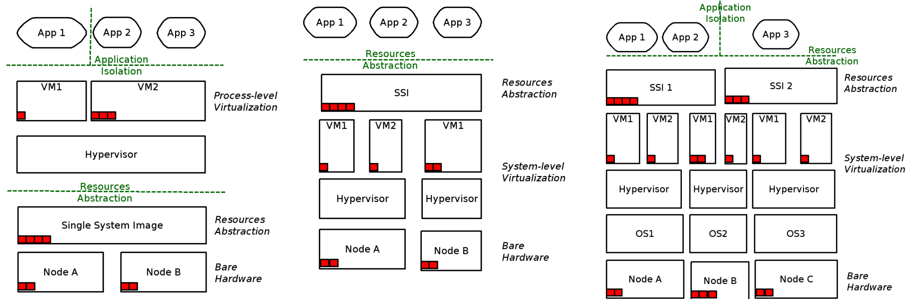


Fig. 5. Type-II Virtualization Upon SSI

Fig. 6. SSI Upon Type-II Virtualization

Fig. 7. Isolation of Two Distinct SSIs

added or removed), and (ii) a *VM live migration* to another node (*e.g.,* the SSI is deployed on top of several VMs, and this number is static). In each case, the property of server consolidation is validated.

Application Portability: Same as AP1, substituting containers by VMs.

Virtual Machine Portability: The type-II virtualization enables the emulation of different hardware architectures. It is therefore possible to migrate VMs to different hardware architectures, only the architecture of the virtual hardware exposed inside the VMs has to be consistent (the SSI does not support heterogeneity). This property is validated.

Suspend/Restart: Same as SR3, substituting type-I by type-II.

5 Lessons

Containers on Top of Single System Image Clusters. Using the container based solution in an SSI, resources exposed to applications can span multiple cluster nodes. By providing the illusion that a cluster is a virtual SMP, the SSI retains all the advantages enabled by containers on a real SMP machine in a cluster environment; removing frontiers between cluster nodes.

Virtual Machines on Top of Single System Image Clusters. This configuration has a major advantage: application portability. For instance, with VMs, it is possible to execute an application developed for processor technology "A" and OS "B" on top of a computer running an SSI OS based on OS "C" and developed for processor technology "D". This means that any application binary can be executed on top of an SSI OS, provided that the appropriate virtualization technology is available. For example, an IIS web server compiled for Windows OS could be deployed on the top of a VM running on an SSI compiled for Linux.

Single System Image on Top of Virtual Machines. Executing an SSI OS on top of a virtual cluster provides a flexible, simple and on demand resource

allocation to applications, but also system-level adaptation in case of cluster configuration changes (node addition and eviction). The idea is to simplify management tasks and to reduce cost of power consumption. If an application requires more (respectively less) resources and more (respectively less) physical cluster nodes, the virtual machines are simply migrated to remote physical nodes. For instance, a multi-threaded Apache server could be deployed on more or less physical nodes according to the amount of requests. Moreover, it becomes possible to execute multiple virtual clusters on the same real cluster, each of them been isolated from the others (for instance, two OpenMP applications can be executed in an isolated way on two different virtual clusters, see Figure 7).

6 Conclusion and Future Works

Nowadays, virtualization technologies are very popular for the execution of applications and services on top of computers. The motivation of this paper was to answer the following question. Do these trends make the SSI for clusters irrelevant for the future?

Based on the current state of the art on SSI and on virtualization techniques, we analysed different configurations combining SSI and virtualization techniques in clusters. From the analysis presented in this paper, we conclude that virtualization and SSI complement each other. A full SSI makes transparent resource distribution in cluster nodes, providing the illusion of a virtual SMP machine (abstraction of distributed resources, see Table 2, cases 4 and 6), whereas the virtualization technologies provide flexibility in resource management (cases 1, 3, and 5).

Table 2. Summary of the different cases studied in this document: (1) Container upon SSI; (2) SSI upon container; (3) type-I upon SSI; (4) SSI upon type-I; (5) type-II upon SSI; (6) SSI upon type-II

	1	2	3	4	5	6
Isolation	-	N/A	+	+	+	+
Server conso.	+	N/A	+	+	+	+
Application Portability	+	N/A	+	+	+	+
VM Portability	-	N/A	-	-	+	+
Suspend/restart	+	N/A	-	+	-	+

We have started experimentations on Kerrighed [13] on top of VMware Server 1.0.4 [7] (no porting effort is required in the current state of the technology). These experiments are realized on a cluster of Grid5000 [16]. We test several kinds of applications: bags of tasks, parallel applications (MPI, OpenMP), servers (Apache with different configurations based on multiple processes or threads). This would allow us to compare the behavior of these applications on clusters running an SSI, running VMs or running one of the combinations of SSI and virtualization solutions that have been identified as attractive. In particular, we plan to measure the applications performance in such environments.

From a more theoretical point of view, we work on designing a model extending the one proposed by Goldberg to present in a uniform framework the hardware, the emulated hardware, the OS, the different virtualization techniques, containers and SSIs.

Hence, we plan to investigate the use of virtualization techniques in a Grid environment for commercial applications requiring strong isolation.

References

1. Grit, L., Irwin, D., Marupadi, V., Shivam, P., Yumerefendi, A., Chase, J., Albrecht, J.: Harnessing virtual machine resource control for job management. In: Proceedings of the First International Workshop on Virtualization Technology in Distributed Computing (VTDC) (November 2006)
2. OpenMosix, http://openmosix.sourceforge.net/
3. Studham, R.S., Cox, A., Walker, B.: Petascale single system image and other stuff (2007)
4. Goldberg, R.P.: Architecture of virtual machines. In: Proceedings of the Workshop on Virtual Computer Systems
5. 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: SOSP 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164–177. ACM, New York (2003)
6. Bellard, F.: Qemu, a fast and portable dynamic translator. Technical report, USENIX Association (2005)
7. VMware: http://www.vmware.com
8. OpenVZ: http://wiki.openvz.org/Main_Page
9. Chroot: ion, http://www.gnu.org/software/coreutils/manual/coreutils.html
10. Ghormley, D.P., Petrou, D., Rodrigues, S.H., Vahdat, A.M., Anderson, T.E.: GLUnix: A Global Layer Unix for a network of workstations. Software Practice and Experience 28(9), 929–961 (1998)
11. Hendriks, E.: BProc: the Beowulf Distributed Process Space. In: ICS 2002: Proceedings of the 16th international conference on Supercomputing, pp. 129–136. ACM Press, New York (2002)
12. Riesen, R., Brightwell, R., Fisk, L.A., Hudson, T., Otto, J., Maccabe, A.B.: Cplant. In: Proceedings of the Second Extreme Linux workshop at the 1999 USENIX Annual Technical Conference (1999)
13. Morin, C., Lottiaux, R., Vallée, G., Gallard, P., Margery, D., Berthou, J.Y., Scherson, I.: Kerrighed and data parallelism: Cluster computing on single system image operating systems. In: Proc. of Cluster 2004. IEEE, Los Alamitos (2004)
14. Barak, A., La'adan, O.: The MOSIX multicomputer operating system for high performance cluster computing. Future Gener. Comput. Syst. 13(4-5), 361–372 (1998)
15. Foundation, A.S.: http://httpd.apache.org
16. Grid5000, http://www.grid5000.fr

Efficient Shared Memory Message Passing for Inter-VM Communications

François Diakhaté[1], Marc Perache[1], Raymond Namyst[2], and Herve Jourdren[1]

[1] CEA DAM Ile de France
[2] University of Bordeaux

Abstract. Thanks to recent advances in virtualization technologies, it is now possible to benefit from the flexibility brought by virtual machines at little cost in terms of CPU performance. However on HPC clusters some overheads remain which prevent widespread usage of virtualization. In this article, we tackle the issue of inter-VM MPI communications when VMs are located on the same physical machine. To achieve this we introduce a virtual device which provides a simple message passing API to the guest OS. This interface can then be used to implement an efficient MPI library for virtual machines. The use of a virtual device makes our solution easily portable across multiple guest operating systems since it only requires a small driver to be written for this device.

We present an implementation based on Linux, the KVM hypervisor and Qemu as its userspace device emulator. Our implementation achieves near native performance in terms of MPI latency and bandwidth.

1 Introduction

Thanks to their excellent isolation and fault tolerance capabilities, virtual machines (VM) have been widely embraced as a way to consolidate network servers and ease their administration. However, virtual machines have not yet been adopted in the context of high performance computing (HPC), mostly because they were incurring a substantial overhead. Recently, advances in hardware and software virtualization support on commodity computers have addressed some of these performance issues. This has led to many studies of how HPC could take advantage of VMs [1].

First, HPC clusters too would greatly benefit from the ease of management brought by virtual machines. Checkpoint/restart and live migration capabilities could provide fault tolerance and load balancing transparently to applications. Moreover, VMs give cluster users a greater control of their software environment without involving system administrators.

Second, it should be possible to use VMs to improve the performance of HPC applications. Using a minimal hypervisor and guest operating system can decrease system noise and therefore greatly increase performance of some communication operations [2]. More generally, virtualization allows the use of specialized guest OSes with scheduling or memory management policies tuned for a specific application class.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 53–62, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Nonetheless, whatever VMs are used for, their integration within HPC environments must have a negligible performance overhead to be successful. Since MPI (*Message Passing Interface*) is the most widely spread communication interface for compute clusters, designing efficient MPI implementations in the context of virtual machines is critical.

Some high performance *Network Interface Cards* can be made accessible directly from within VMs, thus providing near native communication performance when VMs are hosted on different physical machines [3]. However, implementing fast communication between virtual machines over shared memory is also crucial considering the emergence of multicore cluster nodes.

Indeed, to execute MPI applications, it is desirable to use monoprocessor VMs and one MPI task per virtual machine: it allows finer-grained load balancing by VM migration and multiprocessor VMs exhibit additional overheads due to interferences between host and guest schedulers [4,5]. Thus, one major challenge is to implement efficient message passing between processes running inside separate VMs on shared memory architectures.

In this paper, we present a new mechanism to perform efficient message passing between processes running inside separate VMs on shared memory architectures. The main idea is to provide a virtual message passing device that exposes a simple yet powerful interface to the guest MPI library. Our approach enables portability of the guest MPI implementation across multiple virtualization platforms, so that all the complex code dealing with optimizing memory transfers on specific architectures can be reused.

2 Fast MPI Communication over Shared-Memory Architectures

In recent years, shared memory processing machines have become increasingly prevalent and complex. On the one hand, the advent of multi-core chips has dramatically increased the number of cores that can be fitted onto a single motherboard. Intel announcement of an 80 core chip prototype shows that this trend is only going to intensify in the future. On the other hand the increase in the number of sockets per motherboard has led to the introduction of NUMA effects which have to be taken into account. As a result, many research efforts have been devoted to achieving efficient data transfers over these new architectures.

In this section, we give an insight on how to perform such data transfers both in a native context, that is between processes running on top of a regular operating system, and in a virtualized environment (i.e. between processes belonging to separate virtual machines).

2.1 Message Passing on Native Operating Systems

On a native operating system, data exchanges between processes can be performed in many different ways, depending on the size of messages. Nonetheless, all solutions rely only on two primitive mechanisms: using a two step copy through preallocated shared buffers or using a direct memory-to-memory transfer.

Shared memory buffers. Many MPI implementations set up shared memory buffers between processes at startup thanks to standard mechanisms provided by the OS (e.g. mmap, system V shared memory segments). Communication can then take place entirely in userspace by writing to and reading from these buffers. Several communication protocols have been proposed. In MVAPICH2 [6] all processes have dedicated receive buffers for each peer process: no synchronization is required between concurrent sends to a single process, but n^2 buffers are required and the cost of polling for new messages dramatically increases with the number of processes. To alleviate these issues, Nemesis, one of MPICH2 communication channels [7], uses only one receive buffer per process. This requires less memory and ensures constant time polling with respect to the number of processes. However concurrent senders have to be synchronized, which can be done efficiently using lockless queues.

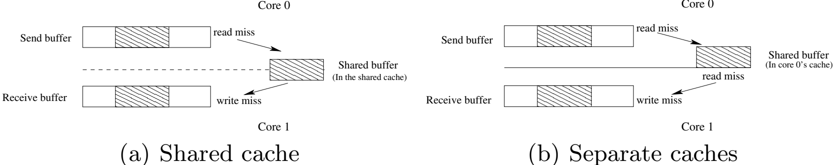
Buffer based communication makes an efficient usage of the shared caches found in multi-core chips because the buffers can be made small enough so that they fit in the cache. As a result, when communicating between two cores sharing a cache, the extra message copy induces very little overhead (Fig. 1). Moreover, high NUMA locality can be achieved by binding each process to a core and allocating their memory buffers on the closest memory node. Such implementations thus offer very low latencies. However, bandwidth usage is not optimal for large messages, especially when processes are executed on cores which do not share a cache level.

Direct transfer. To avoid the extra copy induced by the use of a shared buffer, some MPI implementations aim at directly copying data between send and receive buffers and thus ensure maximum bandwidth usage for large messages. However, this requires that the copy be performed in a memory context where both send and receive buffers are mapped. That for, a first approach is to perform the copy within the operating system's kernel [8]. This requires a dedicated kernel module which makes this solution less portable. Moreover each communication will incur system call or even page pinning costs that restrict usage of this technique to sufficiently large messages. For smaller messages, shared buffer based communication has to be used. Another approach is to use threads instead of processes to execute MPI tasks [9]. This allows a portable, userspace only implementation but requires that the application code be thread-safe. For example, it may not use global variables.

2.2 Message Passing between Virtual Machines on the Same Host

The issues raised when trying to provide shared memory message passing between VMs are actually quite similar to those involved in the native case. Indeed, similarly to processes in an OS, VMs do not share the same virtual address space. Therefore, one of two things is necessary to allow communications between two VMs:

– Having a pool of shared physical pages mapped in both communicating VMs virtual address spaces, so that the VMs can communicate directly using



(a) Shared cache (b) Separate caches

Fig. 1. Performance impact of using a shared buffer: if communicating cores share their cache (a), the additional copy does not lead to a cache miss and therefore causes little overhead compared to a direct copy. On the other hand, if caches are separate (b), significant memory bandwidth is wasted.

any shared memory buffer message passing technique seen earlier with no additional overhead. This approach has been studied using Xen page sharing capability to provide fast socket [10] and MPI [3] communication.

- Requesting a more privileged access to the memory of both VMs to perform the message transfers. This avoids unnecessary copies to a shared buffer but requires a costly transition to the hypervisor.

This is once again a tradeoff between latency and bandwidth. To provide optimal performance, the most appropriate technique has to be picked dynamically depending on the size of the message to transmit. Additionally, VM isolation must be taken into consideration. A VM should not be able to read or corrupt another VM's memory through our communication interface, except for the specific communications buffers to which it has been granted access.

3 Designing a Virtual Message Passing Device

As we pointed out in section 2, two communication channels have to be provided to allow high speed MPI data transfers between VMs running on the same host: a low latency channel based on shared buffers accessible directly from guests' userspaces and a high bandwidth channel based on direct copies performed by the hypervisor.

To address the challenge of providing these two channels to guest OSes in a portable way, we introduce a virtual communication device which exposes a low-level, MPI-friendly interface.

Indeed, traditional operating systems expect to sit directly on top of hardware and to interact with it through various interfaces. Therefore, they already provide ways to deal with these interfaces and to export them to applications. As a result, to introduce guest OS support for a new device one usually only has to write a small kernel module or device driver. Moreover, most hypervisors already emulate several devices to provide basic functionality to VMs (network, block device, etc.) Thus, it is possible to emulate a new device without modifying the core of the hypervisor. A virtual device is therefore an easy and portable way to introduce an interface between hypervisors and guest operating systems.

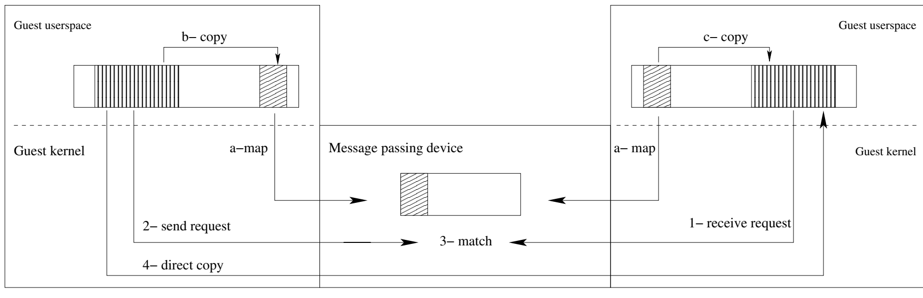


Fig. 2. Overview of the virtual message passing device: shared memory buffers which can be mapped in userspace are provided for low latency communications (a,b,c). Send and receive requests can be posted so that the device performs direct memory copies between VMs (1,2,3,4).

An overview of the usage of these communication channels is provided in Figure 2. We now describe the key features of the device.

Ports. All communication endpoints on a physical machine are uniquely identified by a port number. This allows several communication channels to be opened on each virtual machine to cater for most use cases. Several MPI processes on a single VM might typically want to use the device for communication with MPI processes running in other VMs. Using port numbers instead of VM identifiers allows to handle these cases seamlessly. A VM must open a port before it is able to issue requests originating from it and only one VM can have a given port opened at a time.

Shared memory. The virtual device possesses onboard memory which corresponds to a shared memory buffer that can be used to communicate between VMs. More specifically, this memory is divided into blocks of equal size and each port is attributed a block. The actual communication protocol is unspecified and shall be implemented in the guest's userspace by the communication library (e.g. MPI library). This way, communications can be performed entirely in userspace and don't incur latency overheads due to context switching.

DMA transfers. The virtual device can process DMA copy operations between arbitrary memory locations on all the VMs that use it. There are 2 types of requests: *receive* and *send* which both apply to an origin and a target port. They take two additional arguments:

– A list of pointers and sizes which describe a possibly scattered memory buffer to send to or receive from.
– A completion register which is a pointer to an integer in the VM's memory. Completion and failure of a request can be inferred by reading the specified

integer. This allows to poll directly from the guest's userspace, thus reducing the number of transitions to the guest kernel and to the hypervisor.

The semantics of these operations are similar to MPI_Irecv and MPI_Isend semantics. In particular, they ensure that a VM can only write to an other VM's memory when and where it is authorized to do so.

Note that this interface is similar to those of high performance network cards which offer buffered and rendez-vous communication channels. This ensures that existing MPI libraries can be ported easily to support this virtual messaging device. While our solution still requires more porting work than a solution offering binary compatibility with the socket interface [10] it is also more efficient, if only because it doesn't incur the system call overhead induced by sockets.

4 Implementation

In this section we provide details on our preliminary implementation of this virtual device using Linux as both guest and host OS and the Kernel Virtual Machine hypervisor. We start by providing a brief overview of KVM and then proceed to describe the implementation of our device from the guest and host point of view.

4.1 The Kernel Virtual Machine

KVM is a Linux kernel module which allows Linux to act as an hypervisor thanks to hardware virtualization capabilities of newer commodity processors. It is thus capable of virtualizing unmodified guest OSes but also supports paravirtualization to optimize performance critical operations. It provides paravirtualized block and network devices and a paravirtualized MMU interface.

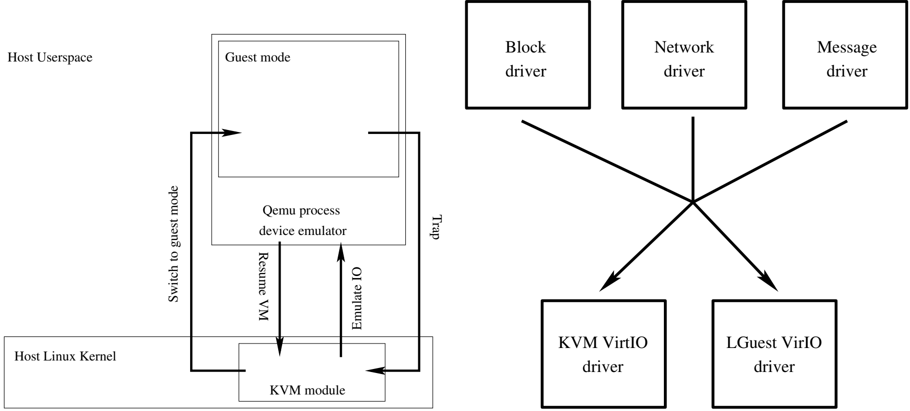


Fig. 3. Architecture of the KVM hypervisor

Fig. 4. The VirtIO interface

Using KVM, VMs are standard Linux processes which can be managed and monitored easily. A userspace component is used to perform various initialization tasks, among which allocating a VM's memory using standard allocation functions and launching it by performing an *ioctl* call on the KVM module (see Figure 3). This component is then asked to emulate the VM's devices: whenever the guest OS tries to access the emulated device, the corresponding instruction traps into the KVM module which forwards it to the userspace component so that the expected result may be emulated. A slightly modified version of QEMU is provided as KVM's default userspace component as it is capable of emulating many devices.

As a result, it is interesting to note that in this model, implementing the emulation of a new virtual device does not introduce additional code into the host kernel. Everything is performed inside QEMU which is a userspace process.

4.2 Guest Implementation

A Linux driver has been implemented to allow Linux guests to handle our device. In the following paragraphs, we describe how this driver accesses the device and how it exposes the virtual device's functions to userspace applications.

Accessing the device. To maximize portability, our device is accessed through the recently introduced VirtIO interface. This interface abstracts the hypervisor specific virtual device handling instructions into an hypervisor agnostic interface. As a result, as shown in Figure 4, the same drivers can be used to handle several hypervisors' implementations of a given device. Only one hypervisor specific driver is needed to implement the interface itself. The VirtIO interface is based on buffer descriptors, which are lists of pointers and sizes packed in an array. Operations include inserting these descriptors into a queue, signalling the hypervisor and checking if a buffer descriptor has been used.

In our virtual device, each open port is associated with a VirtIO queue through which DMA requests can be sent by queuing buffer descriptors. The first pointer/size pair of the buffer descriptor points to a header containing the parameters of the request: whether it is a send or a receive, its buffer size, its destination port and its completion register. The following pointer/size pairs describe the buffer to send or receive.

However there is no VirtIO interface for directly sharing memory between guests. Therefore the shared memory is exposed to guests as the onboard memory of a PCI device.

Userspace interface. Our Linux driver exposes the virtual device as a character device and uses file descriptors to define ports. A port is acquired with the *open* system call, which returns the corresponding file descriptor and released with *close*. *Ioctl* calls are used to issue DMA requests because they require custom arguments such as the destination port and completion register. *Mmap* allows to map the shared memory of the device in userspace.

4.3 Host Implementation

The host is in charge of emulating the virtual device. It has to implement memory sharing and DMA copies between VMs. Using KVM, everything can be implemented inside the VMs' corresponding QEMU processes as described below.

Memory sharing between QEMU instances. Since each VM's device emulation is performed by its own QEMU process these processes need to share memory to communicate: DMA requests passed through VirtIO queues must be shared so that send and receives may be matched and each QEMU instance needs to be able to access any VM's memory to perform the copies between send and receive buffers. Moreover, a shared memory buffer must be allocated and mapped as the onboard memory of a PCI device to expose it to guests.

Our QEMU instances are slightly modified so that they allocate all of this memory from a shared memory pool. This is currently achieved by allocating this memory pool in one process with mmap and the MAP_SHARED flag before creating the QEMU instances for the VMs. These instances are then created by forking this initial process.

One limitation of this implementation is that it cannot support a varying number of communicating VMs. However we plan to support this by allocating each QEMU instance's memory separately using a file backed mmap. QEMU instances would then be able to access each other's memories by mapping these files into their respective address spaces. Another possibility would be to run VM instances inside threads of a single process but this would require to make QEMU thread-safe.

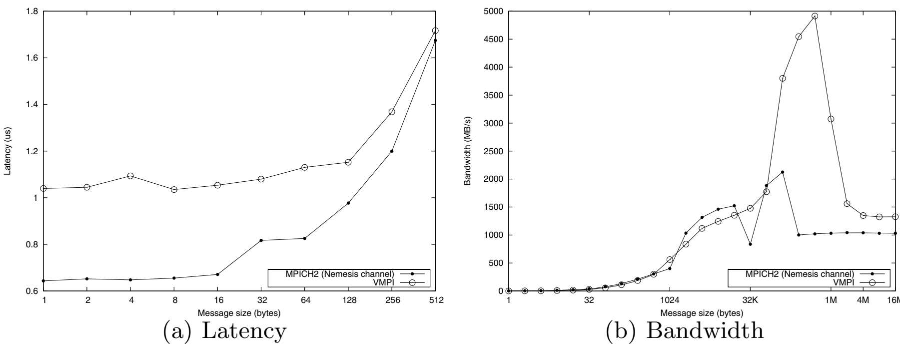
DMA transfers. Whenever a guest signals that it has queued DMA requests through the VirtIO interface, the corresponding QEMU instance will dequeue and process these requests. Receives are stored in a per port queue which is shared among QEMU instances and for each send, the corresponding receive queue is searched for a matching receive. If one is found, a copy is performed between the send and receive buffers and the completion registers are updated.

5 Evaluation

To evaluate our virtual device implementation, we have developed a minimal MPI library which provides MPI_Irecv, MPI_Isend and MPI_Wait communication primitives. It should be noted that most other MPI calls can be implemented on top of these basic functions.

Small messages ($\leq$ 32 KB) are transmitted over the shared buffers provided by the virtual device. This 32KB limit has been determined empirically. Each MPI task receives messages in the shared memory block of the virtual port it has been attributed. This memory block is used as a producer/consumer circular buffer and is protected by a lock to prevent concurrent writes.

For larger messages, we use a rendez-vous protocol. The sender sends the message header to the shared buffer of the receiver. In turn, when the receiver



(a) Latency (b) Bandwidth

Fig. 5. Results

finds a matching receive, it posts a DMA receive request to the virtual device. It then sends an acknowledgment to the sender which can post a DMA send request.

We evaluate the performance of our implementation on a pingpong benchmark. We measure the latency and bandwidth achieved between two host processes communicating using MPICH2 and between two processes on two VMs using our virtual device and minimal MPI implementation (VMPI). The machine used for this test has two quad-core Xeon E5345 (2.33Ghz) processors and 4GB of RAM. Processes or VMs are bound to different processors. Results are shown in Figure 5.

Nemesis uses lockless data structures and minimizes the number of instructions on the critical path which explains its better performance in small message latency compared to our naive MPI implementation. By implementing Nemesis' communication algorithm in our MPI library we should be able to attain similar performance. The bandwidth graph shows that as long as we use the virtual device's shared buffers for communication (up to 32 KB), there are little difference between the native and virtualized case. For larger messages, the ability to perform direct copies between send and receive buffers allows us to outperform Nemesis even when messages don't fit in the cache. This result outlines an interesting property of virtualization: it can be used to implement optimizations that cannot be performed natively without introducing privileged code.

6 Conclusion and Future Work

In this paper, we presented the design and implementation of a virtual device for efficient message passing between VMs which share the same host. Our evaluation shows that it achieves near native performance in MPI pingpong tests and can outperform native userspace MPI implementations without introducing privileged code on the host.

In the future, we intend to integrate support for our device as a channel in an existing MPI implementation so as to provide the whole MPI interface. This will

allow a more thorough evaluation of our solution on real HPC applications. We also plan to extend our virtual device so that it supports additional features such as live migration. This will require a callback mechanism to allow the MPI library to suspend and resume communications appropriately when tasks are migrated. On an other note, we plan to experiment with specialized VM scheduling policies that take communications and NUMA effects into account to reduce overheads when there are more VMs than available cores.

References

1. Mergen, M.F., Uhlig, V., Krieger, O., Xenidis, J.: Virtualization for high-performance computing. SIGOPS Operating Systems Review 40(2) (2006)
2. Hudson, T., Brightwell, R.: Network performance impact of a lightweight linux for cray xt3 compute nodes. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, SC 2006 (2006)
3. Huang, W., Koop, M., Gao, Q., Panda, D.K.: Virtual Machine Aware Communication Libraries for High Performance Computing. In: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, SC 2007 (2007)
4. Ranadive, A., Kesavan, M., Gavrilovska, A., Schwan, K.: Performance implications of virtualizing multicore cluster machines. In: 2nd Workshop on System-level Virtualization for High Performance Computing, HPCVirt 2008 (2008)
5. Uhlig, V., LeVasseur, J., Skoglund, E., Dannowski, U.: Towards scalable multiprocessor virtual machines. In: Proceedings of the 3rd Virtual Machine Research and Technology Symposium, VM 2004 (2004)
6. Chai, L., Hartono, A., Panda, D.K.: Designing high performance and scalable mpi intra-node communication support for clusters. In: Proceedings of the 2006 IEEE International Conference on Cluster Computing, Cluster 2006 (2006)
7. Buntinas, D., Mercier, G., Gropp, W.: Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem. In: Proceedings of the 13th European PVM/MPI Users Group Meeting, EuroPVM/MPI 2006 (2006)
8. Jin, H.W., Panda, D.K.: Limic: Support for high-performance mpi intra-node communication on linux cluster. In: Proceedings of the 2005 International Conference on Parallel Processing, ICPP 2005 (2005)
9. Demaine, E.D.: A threads-only mpi implementation for the development of parallel programs. In: Proceedings of the 11th International Symposium on High Performance Computing Systems, HPCS 1997 (1997)
10. Kim, K., Kim, C., Jung, S.I., Shin, H.S.: Inter-domain socket communications supporting high performance and full binary compatibility on xen. In: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, VEE 2008 (2008)

An Analysis of HPC Benchmarks in Virtual Machine Environments*

Anand Tikotekar, Geoffroy Vallée, Thomas Naughton, Hong Ong, Christian Engelmann, and Stephen L. Scott

Oak Ridge National Laboratory
Computer Science and Mathematics Division
Oak Ridge, TN 37831, USA
{tikotekaraa,valleegr,naughtont,hongong,engelmannc, scottsl}@ornl.gov

Abstract. Virtualization technology has been gaining acceptance in the scientific community due to its overall flexibility in running HPC applications. It has been reported that a specific class of applications is better suited to a particular type of virtualization scheme or implementation. For example, Xen has been shown to perform with little overhead for compute-bound applications. Such a study, although useful, does not allow us to generalize conclusions beyond the performance analysis of that application which is explicitly executed. An explanation of why the generalization described above is difficult, may be due to the versatility in applications, which leads to different overheads in virtual environments. For example, two similar applications may spend disproportionate amount of time in their respective library code when run in virtual environments. In this paper, we aim to study such potential causes by investigating the behavior and identifying patterns of various overheads for HPC benchmark applications. Based on the investigation of the overhead profiles for different benchmarks, we aim to address questions such as: Are the overhead profiles for a particular type of benchmarks (such as compute-bound) similar or are there grounds to conclude otherwise?

1 Introduction

Increasingly, HPC applications are being deployed on virtual environments such as Xen. The reason for such a trend is that the flexibility provided by virtual environments, such as the ability to facilitate fault-tolerance, could balance any performance costs. Indeed, many studies [11] [6] [4], have indicated that performance penalty arising from virtualization schemes is not significant. Furthermore, research has established that I/O bound applications incur more performance penalty on Xen than compute-bound applications [6]. Yet, we cannot generalize such a performance conclusion even for similar applications only on the basis of a final performance number. For example, it is possible that two different performance overhead profiles may ultimately post a similar performance penalty, but for different reasons. Thus, the problem of predicting performance

* ORNL's work was supported by the U.S. Department of Energy, under Contract DE-AC05-00OR22725.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 63–71, 2009.
© Springer-Verlag Berlin Heidelberg 2009

for applications is difficult, and becomes even more difficult in virtual environments due to its complexity.

The complexity inherent in virtual environments can lead to unpredictable application performance such as incurring disproportionate overhead when code contribution is changed (for example, increase in the user code may be impacted disproportionately). Further, the impact of events such as ITLB, DTLB, and cache misses can also contribute towards the difficulty of performance prediction due to an indirection layer of virtualization. But the problem can be alleviated by studying the details of the impact of virtualization on applications.

Understanding the details about the impact of virtualization on HPC application is useful for the following reasons: First, it can uncover an application's behavior in virtual environments. Second, it allows us to identify sources of various overhead costs. Third, its possible that two applications may have similar gross performance numbers, but the composition of performance penalty may be totally different. Fourth, by analyzing the impact of virtualization, we can state more confidently whether we can generalize the performance conclusions.

Our primary objective in this paper is to study the impact of Xen on the behavior of HPC applications in detail. In particular, we compare the impact of Xen [4] on HPCC [1] and NPB [2]. In the process, we study how Xen affects various parts of HPCC and NPB.

The organization of our paper is the following: Section 2 presents related work in the area. Section 3 describes the settings used to study the analysis of the impact of Xen on HPC application behavior profiles. In Section 4, we detail our results based on two HPC application profiles. In Section 5, we discuss and analyze our results. In section 6, we present our conclusion and future work.

2 Related Work

Xenoprof [8] is one of the few tools that can be used as a system wide profiler on Xen. Xenoprof was used to diagnose performance overheads in network applications. The authors also study various events such as L2cache misses, ITLB misses, and correlate them in their study. However, the original goal was to identify performance bugs using the data collected by Xenoprof/OProfile.

The TLB behavior for scientific applications on commodity microprocessors was studied in [7]. Their work is similar in theme to ours. Their conclusion is that while SPEC CPU and HPCC benchmark suits represent cache behaviors of the high-end scientific applications, they fall short when it comes to TLB behavior, and thus can have significant performance consequences. In this paper, we want to emphasize the difficulty of generalizing performance conclusions in virtual environments.

Work in [10] studies memory hierarchy characteristics of para-virtualized systems. The authors also study hardware counters using Xenoprof for memory intensive applications such as DGEMM. The authors conclude than Xen provides near native execution performance and similar memory hierarchy profiles. Our work attempts to compare impacts of Xen on two HPC applications in order to study their profiles.

Work reported in [3] points out that there is a need to consider real HPC applications for performance evaluations and benchmarking. The authors also compare

performance results from kernel benchmarks to the real-world applications, and find that kernel benchmarks do not fully represent real world scientific applications.

Other studies such as [5] [11] concluded that Xen impacts HPC applications minimally. Our study extends previous work by attempting to determine if we can generalize such conclusions beyond those applications that are expressly studied.

3 Evaluation Methodology

In this section, we outline the experimental settings used to gather results and perform post-analysis.

3.1 Applications

We have used the HPCC and NPB application benchmark suites for our study. HPL and SP are used as work-loads to study the compute-bound properties of an application. The problem sizes are 6000 and 162 (class C) for HPL and SP respectively.

3.2 Native and Virtual Machine Environments

Our system environment consists of a 16 node cluster. Each node has a 2Gz Pentium 4 processor, 768MB of RAM, and a 256KB L2 cache, connected by a 100Mb Ethernet switch. Our "Native" environment consists of a Linux 2.6.16.33 kernel with the Fedora Core 5 (FC5) filesystem distribution. Our "Virtual Machine" environment runs on Xen 3.0.4, Linux kernel 2.6.16.33, with 512MB of memory for each virtual machine with one virtual machine per node. We use the same filesystem as that of Native for HostOS. The filesystem for a virtual machine is a disk based flat file of 2GB using FC5. We use a NFS shared filesystem on all three platforms.

3.3 Profiling and Data Collection Tools

We use Oprofile 0.9.1 as our data gathering tool. Oprofile is a system wide statistical profiler. Oprofile uses CPU counters to generate events based on a configurable frequency, which we have set to 100000. This frequency instructs Oprofile to generate a sample for every 100000 occurrences of a specific configurable event such as DTLB miss and attribute it to the code that caused the counter associated with that event to overflow.

We study four events: clock-unhalted, ITLB miss, DTLB miss, and L2Cache miss. For each event, we gather the breakdown of the samples of an application into various parts such as application code, library code, kernel modules, kernel code, and hypervisor code. The clock-unhalted event is a measure of CPU processing time. ITLB and DTLB miss events measure the time spent by the page walk handler. The L2cache miss event is a read level cache miss.

Custom scripts [9] were developed to parse the collected data into application code, library code, kernel modules, kernel code, and hypervisor code.

4 Performance Evaluation

4.1 Overall Penalty

Table 1 shows us the overall performance penalty on HostOS (which is the Dom0 virtual machine) as well as on VMs in terms of the wall clock time, the number of samples, and the instructions executed. The table shows that the overhead in number of samples in virtual environments (at least HostOS, but possibly VMs too, as explained below) is more compared to the wall clock time overhead. One explanation being that: even though the clock-unhalted event, as described earlier, is a measure of CPU processing time, it is a measure of time when the CPU is active. Therefore, when the CPU is idle, as when there is an I/O or memory transfer, this event is not useful. Further, the CPU executes a fewer number of instructions on native compared to virtual environment as shown by Table 1, and thus can remain idle longer than say HostOS, which can execute more instructions in parallel to I/O. Please note that, because of Xen's architecture [4], the samples for a VM are split into DomU and Dom0. The application executes in DomU and therefore contains the bulk of the samples, but Dom0 also contains part of the application samples when the application requires backend device drivers (such as for performing I/O) located in the Dom0 HostOS kernel. Further, also note that, even though the application samples for a VM are located in DomU and Dom0, we only analyze the DomU side of the samples in the case of VMs because of a known limitation of Xenoprof, which does not allow us to isolate samples from Dom0 profile which are part of the applications running in DomU. Therefore, in the following analysis, we indicate Dom0 samples by greek letters such as δ and γ. And unless otherwise stated, when we refer to the VM, we mean the DomU portion of the Virtual Machine.

4.2 Breakdown of Overall Penalty

Figure 1 shows the breakdown of the time spent by each application into its various parts across native, HostOS and VM environments. Overhead cost of user code in HPL

Table 1. Performance penalty as compared to native

	HostOS penalty %			VM penalty %		
	Wall clock time	No. of samples	Instructions executed	Wall clock time	No. of samples	Instructions executed
HPCC- HPL	2	8	2	12	11 + δ	5
NPB - SP	1	5	9	18	9 + γ	11

Table 2. Breakdown of performance penalty for clock samples as compared to native - δ_{clk} and γ_{clk}: Dom0 part of HPL and SP respectively

	HPL-App	SP-App	HPL-Lib	SP-Lib	HPL-Sys	SP-Sys
HostOS penalty%	5	1	6	138	50	49
VM penalty %	13	4	8	118	88 + δ_{clk}	62 + γ_{clk}

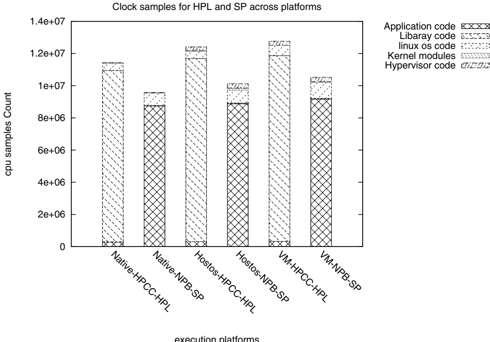


Fig. 1. Comparison of breakdown of CPU samples for HPL and SP across platforms - Results for VM do not contain Dom0 samples

is more than that of SP. One reason is that user code contribution is more in HPL than in SP, and therefore virtualization impacts it disproportionately. Interestingly, the overhead cost of system code under HostOS and VM in SP is less than that of HPL even though SP spends twice as much time in system code as HPL on native. This can be because HPL spends proportionately more time in hypervisor code than SP does. Furthermore, the overhead costs are more when applications are running in virtual machines than when they are running in the HostOS.

Table 2 shows how various parts of HPL and SP are being impacted differently in virtual environments. The most obvious is the small contribution from SP's library code. Since the library code of SP only forms a small fraction of the overall code distribution, its impact in virtual environments does not show up in Figure 1. Similarly, the system code is expensive in virtual environments even though it may not be apparent in Figure 1 as the system code is only 10% of the overall code. The system code penalty distribution among *kernel modules*, *kernel core* and *hypervisor* on the HostOS are: 9%, 62%, 29% for HPL, and 10%, 66%, 24% for SP. Similarly, the penalty distribution for system code under the VM (DomU only) is *kernel core* and *hypervisor* are: 72%, 28% for HPL, and 77%, 23% for SP. Note, the contribution of kernel modules under VMs is part of Dom0 and therefore not shown as explained previously (Section 4.1).

Further, system code penalty for HPL on VMs is more than that of SP. One explanation is that HPL code performs more privileged operations than SP. The reason why the impact of Xen on the library code in SP is so drastic compared to HPL is unclear and may additionally require sophisticated tracing to diagnose the problem. Thus, while the overall performance penalty is only one number, Table 2 shows us the actual "behind-the-scene" story. . In light of this information, it is difficult to generalize the performance conclusions to other applications. In the next few sections, we study other events such as ITLB miss, DTLB miss and L2 cache miss.

4.3 Breakdown of DTLB Miss Samples

Figure 2 shows the comparison of DTLB misses across platforms for our two applications. From the figure, we can see that the impact of virtualization is limited to the

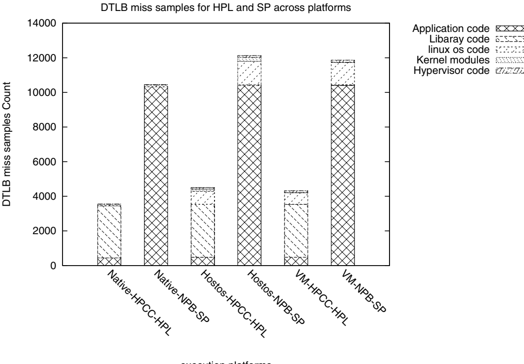


Fig. 2. Comparison of breakdown of DTLB Miss samples for HPL and SP across platforms - Results for VM do not contain Dom0 samples

Table 3. Breakdown of performance penalty for DTLB miss samples as compared to native

	HPL-App	SP-App	HPL-Lib	SP-Lib	HPL-Sys	SP-Sys
HostOS penalty%	7	0.6	1	1900	800	1300
VM penalty %	7	0.6	1.6	1500	700 + δ_{dtlb}	1150 + γ_{dtlb}

system side. Further, by looking at Figure 1, one might conclude that Xen impacts HPL's library code more than that of SP's library code. But as described in our previous section, the contribution of the library code in SP is very small and therefore does not show up in Figure 1. However, Table 2 shows that the library code in SP is impacted drastically, and is supported by the fact that the DTLB miss rate increases for SP's library code, and remains very low for HPL as shown in Table 3. The huge performance penalty numbers like 1900% arise because the number of DTLB miss samples increases from 3 to 60. The story for the system side described by Figure 1 is also supported by Figure 2, in that DTLB rate increases for both HPL and SP, although in different ways. The impact of Xen on DTLB miss rate is more for SP's system code than HPL's under HostOS and VM. As stated before, we cannot comment on the δ_{dtlb} and γ_{dtlb} from Dom0.

4.4 Breakdown of ITLB Miss Samples

Figure 3 shows the comparison of ITLB misses across platforms for our two applications. Figure 3 and Table 4 show that Xen impacts the system side more than the user side but the impact is not limited to the system side. First, the ITLB miss rate continues to support the fact that Xen does impact SP's library code drastically. Second, the ITLB miss rate (Table 4) is also consistent with Figure 1 in that it partly explains why Xen impacts HPL's system code more than SP's under both HostOS and VM. Yet, as shown in Table 3, the DTLB miss rate does not explain why Xen impacts HPL's system side more than SP's. Moreover, one can easily see that Table 3 and Table 4 support Table 2 when it comes to application-only code.

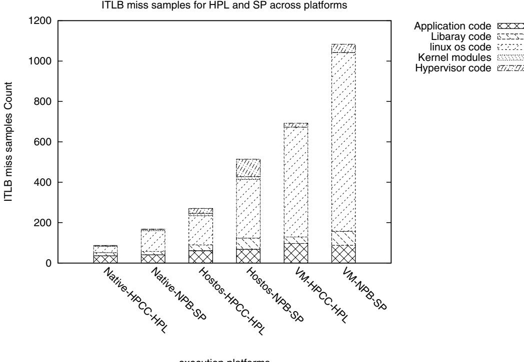


Fig. 3. Comparison of breakdown of ITLB miss samples for HPL and SP across platforms - Results for VM do not contain Dom0 samples

Table 4. Breakdown of performance penalty for ITLB miss samples as compared to native - δ_{itlb} and γ_{itlb}: Dom0 part of HPL and SP respectively

	HPL-App	**SP-App**	**HPL-Lib**	**SP-Lib**	**HPL-Sys**	**SP-Sys**
HostOS penalty %	74	70	74	243	417	257
VM penalty %	177	117	93	331	1500 + δ_{itlb}	750 + γ_{itlb}

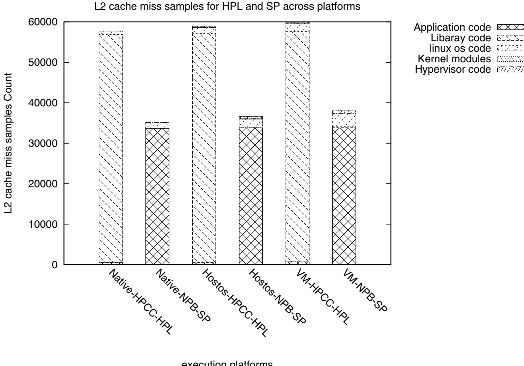


Fig. 4. Comparison of breakdown of L2 miss samples for HPL and SP across platforms - Results for VM do not contain Dom0 samples

4.5 Breakdown of L2 Cache Miss Samples

The comparison of L2 cache miss samples is shown in Figure 4. Table 5 shows that the impact of Xen on L2 cache miss samples is restricted to system side only, except SP's library code. Table 5 shows mixed results. The L2 cache miss rate for the system code on the HostOS is greater for HPL than SP, and on VMs it is the opposite, SP being greater than HPL.

Table 5. Breakdown of performance penalty for L2 cache samples as compared to native - δ_{l2} and γ_{l2}: Dom0 part of HPL and SP respectively

	HPL-App	SP-App	HPL-Lib	SP-Lib	HPL-Sys	SP-Sys
HostOS penalty%	10	0.2	0.4	130	104	101
VM penalty %	30	0.7	0.9	500	171 + δ_{l2}	186 + γ_{l2}

5 Discussion

Our previous section establishes that HPL and SP are impacted differently by Xen. As shown in our study, the applications have different characteristics even though both are compute bound. This supports our premise that we can not generalize performance in virtual environments. A detailed analysis is useful to understand the application workload. For instance, HPL spends most of its user code time in the BLAS library, while most of the user code in SP is located in the application itself. Second, SP has a greater contribution from system code than HPL has from its system code.

The conclusion that the impact of Xen is mainly restricted toward the system code and not the user code is accepted based on Xen's para-virtualized architecture. However, this paper has indicated that while Xen impacts system code much more than user code, there is evidence, such as in the case of SP's library code, that the user code may not be immune from Xen's impact.

6 Conclusion

We have studied and analyzed HPL and SP from HPCC and NPB respectively. Our goal for the study was to determine the impact of Xen on these applications and compare the penalty profiles of these two applications. It is important to note that we are not only concerned with the "final performance penalty" number but the composition that makes up the overall performance penalty.

We found that, while the overall performance penalty does not differ much between HPL and SP, their overhead profiles are not similar. Further, we found that Xen impacts the various parts of these applications in different ways. It is therefore possible that different applications in the same class may be impacted more differently than HPL or SP.

We also found that the similar final performance impact of HPL and SP is not entirely due to the fact that these are compute-bound benchmark applications, but because the parts that are impacted differently by Xen are too small to influence the final performance number.

Our findings emphasize the difficulty of performance prediction and generalization. Moreover, as we have seen, performance isolation, especially on VMs remains difficult to achieve.

We plan to extend our study to more scientific applications. We would like to determine whether similar benchmark applications have versatility such that Xen impacts them differently or not. Further,We would like to work on the limitations of the performance measurement tools, such as Xenoprof, so that we can enhance application profiling.

References

1. HPC challenge, `http://icl.cs.utk.edu/hpcc`
2. NAS parallel benchmarks, `http://www.nas.nasa.gov/Resources/Software/npb.html`
3. Armstrong, B., Baeh, H., Eigenmann, R., Saied, F., Sayeed, M., Zheng, Y.: HPC benchmarking and performance evaluation with realistic applications. In: 2006 SPEC Benchmark Workshop, SPEC (2006)
4. 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 (SOSP19), pp. 164–177. ACM Press, New York (2003)
5. Emeneker, W., Stanzione, D.: HPC Cluster Readiness of Xen and User Mode Linux. In: IEEE International Conference on Cluster Computing (September 2006)
6. Huang, W., Liu, J., Abali, B., Panda, D.K.: A Case for High Performance Computing with Virtual Machines. In: 20th ACM International Conference on Supercomputing (ICS 2006), Cairns, Queensland, Australia (June 2006)
7. McCurdy, C., Cox, A., Vetter, J.: Investigating the TLB behavior of high-end scientific scientific applications on commodity microprocessors. In: IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2008 (2008)
8. Menon, A., Santos, J.R., Turner, Y., Janakiraman, G., Zwaenepoe, W.: Diagnosing performance overhead in the Xen virtual machine environment. In: Proceedings of the 1st ACM Conference on Virtual Execution Environments (June 2005)
9. Tikotekar, A., Vallee, G., Naughton, T., Ong, H., Engelmann, C., Scott, S.L.: Effects of virtualization on a scientific application. In: 2nd Workshop on System-level Virtualization for High Performance Computing (HPCVirt 2008) held in conjunction with EuroSys (2008)
10. Youseff, L., Seymour, K., You, H., Dongarra, J., Wolski, R.: The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software. In: ACM/IEEE International Symposium on High Performance Distributed Computing, HPDC (2008)
11. Youseff, L., Wolski, R., Gorda, B., Krintz, C.: Paravirtualization for HPC systems. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 474–486. Springer, Heidelberg (2006)

UNICORE Summit 2008

The UNICORE grid technology provides a seamless, secure and intuitive access to distributed grid resources. UNICORE is a full-grown and well-tested grid middleware system, which today is used in daily production worldwide. Beyond this production usage, the UNICORE technology serves as a solid basis in many European and International projects. In order to foster these ongoing developments, UNICORE is available as open source under BSD licence at http://www.unicore.eu.

The UNICORE Summit is a unique opportunity for grid users, developers, administrators, researchers and service providers to meet. The First UNICORE Summit was held in conjunction with "Grids@work - 2nd Grid Plugtests," October 11 – 12, 2005 in Sophia Antipolis, France. In 2006 the style of the UNICORE Summit was changed by establishing a Program Committee and publishing a Call for Papers. The UNICORE Summit 2006 was held in conjunction with the Euro-Par 2006 conference in Dresden, Germany, August 30 – 31, 2006. The proceedings are available as LNCS volume 4375. The UNICORE Summit 2007 was held in conjunction with the Euro-Par 2007 conference in Rennes, France, on August 28, 2008. The proceedings are available as LNCS volume 4854.

In 2008 the UNICORE Summit was held again in conjunction with the Euro-Par conference, this time in Las Palmas de Gran Canaria, Spain, on August 26.

We would like to thank the Program Committee members Agnes Ansari, Rosa Badia, Thomas Fahringer, Donal Fellows, Anton Frank, Edgar Gabriel, Alfred Geiger, Frerik Hedman, Odej Kao, Paolo Malfetti, Ralf Ratering, Mathilde Romberg, Bernd Schuller, Dave Snelling, Thomas Soddemann, Stefan Wesner and Ramin Yahyapour for their excellent job. Special thanks go to Max Berger and Kassian Plankensteiner for providing additional reviews.

Finally, we would like to thank all authors for their submissions, camera-ready versions and presentations at the UNICORE Summit 2008 in Las Palmas de Gran Canaria as well as Emilio Benfenati for giving the opening talk.

Most likely, the next UNICORE Summit will again take place in conjunction with the Euro-Par conference. More information can be found at http://www.unicore.eu/summit. We are looking forward to the next UNICORE Summit !

October 2008

Achim Streit
Wolfgang Ziegler

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, p. 73, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Space-Based Approach to High-Throughput Computations in UNICORE 6 Grids

Bernd Schuller and Miriam Schumacher

Jülich Supercomputer Centre
Distributed Systems and Grid Computing Division
Forschungszentrum Jülich GmbH
Jülich, Germany

Abstract. We explore a novel approach to high-throughput, embarassingly parallel applications in UNICORE 6 based Grids. This is an XML centric tuple space based approach inspired by JavaSpaces, with an implementation using the UNICORE 6 WSRF framework. Other approaches such as the layered workflow and data splitting architecture developed in the Chemomentum project, batch processing using the UNICORE commandline client UCC and concurrent programming using the HiLA Java API are discussed as well. Performance and scalability evaluations are presented, backed up by preliminary experimental results comparing our approach to the standard batch mode of the UNICORE commandline client.

1 Introduction

In this paper we explore an approach to high-throughput computing on UNICORE Grids that is based on an XML tuple space, i.e. a globally shared storage, that offers a simple API for storing,reading and removing arbitrary XML documents. The term high-throughput computing is used here in the sense that many relatively small computational jobs are run on a Grid system composed of relatively many compute nodes. To give an example, a typical run might involve several thousands of jobs, where each job consumes about 15 minutes of computational time, and with a number of compute nodes that is on the order of 100. For the purposes of this paper, the application throughput is assumed to be limited by the computation itself, not by the associated data transfers. High-throughput computing as defined here is highly relevant in many application fields from drug discovery to multi-media applications such as image or video rendering. A prominent example is in-silico screening of chemical substances using docking techniques, for example in the WISDOM initiative [2]. The main problems to be overcome are scalability, efficient resource discovery, selection and usage. Conventional approaches to resource discovery and selection involve information systems, where the resource providers have to publish detailed information about the state of their resources. This is often undesirable, as this information may be confidential. It is difficult to keep this information up-to-date, and the information system may become the bottleneck.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 75–83, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Additionally, sites may have to give up some of their autonomy to allow efficient resource management by the Grid scheduling systems. As will be shown, the tuple space-based approach removes the need for these information systems, and the sites can trivially enforce their local policies.

The remainder of the paper is organised as follows. Section 2 introduces the UNICORE 6 Grid middleware. Existing approaches to high-throughput computing using UNICORE are summarised in section 3, while our tuple space-based approach is introduced in section 4, Some preliminary performance results are given in section 5. A summary and outlook concludes the paper.

2 The UNICORE 6 Grid Middleware

UNICORE, developed in the course of several German and European projects since 1997 [1], is a mature Grid middleware that is deployed and used in a variety of settings, from small projects to large (multi-site) infrastructures involving high-performance computing resources. UNICORE can be characterised as a vertically integrated Grid system, that comprises the full software stack from clients to various server components down to the components for accessing the actual compute or data resources. Its basic principles are abstraction of site-specific details, openness, interoperability, operating system independence, security, and autonomy of resource providers. In addition, the software is easy to install, configure and administrate. The latest version is UNICORE 6 [4], which is based on Web Services and particularly the Web Service Resource Framework (WSRF). UNICORE is licensed under the liberal BSD license, and is available as open source from the SourceForge repository [3].

UNICORE 6 is a four-tiered system, consisting of the client, gateway, services and target system tiers. A wide variety of clients exist, from programming APIs [14], commandline client [15], simple Java clients to a rich client based on the Eclipse framework. The Gateway is a thin authentication and routing service that can be considered as a web service firewall and router. It resides outside the networking firewall, protecting the services behind it. Thus, UNICORE by default only requires a single open port to the public internet. The basic services (UNICORE atomic services) provide resource discovery (Registry service), job execution (Target System Factory and Target System services), and file access (Storage and FileTransfer services).

The target system tier consists of the interface to the local operating system, file system and resource management (batch) system. UNICORE 6 uses XML based standards in all functional areas: WS(RF)/SOAP for communication, JSDL for job submission, SAML assertions and XACML for authentication and authorisation.

The UNICORE 6 service registry contains the available target system factory (TSF) services, not the target system services (TSS) themselves. The reason for this is that each client (i.e. Grid user) creates their own target systems, which are accessible only for that particular client. The TSFs keep a list of TSS created by them. Thus, the discovery of available target systems is a fairly expensive

operation involving several web service calls, because clients have to iterate over these TSS lists, and check for accessible services.

The basic sequence to run a computational job on UNICORE 6 is as follows.

– A suitable computational resource (target system service, TSS) needs to be found. If no TSS is available to the client, a suitable target system factory (TSF) must be discovered and invoked to create a TSS
– The job is submitted to the TSS, resulting in a new job management service (JMS) instance
– Input data can be staged in to the job's working directory
– The job is started. Usually the client sends a "start" message to indicate it has finished staging data in.
– After the job finishes, output data can be staged out.

At the time of writing (April 2008), UNICORE does not support client notification on job status changes, so a polling approach has to be used to find out if a job has finished.

3 High-Thoughput Approaches for UNICORE 6

3.1 Batch Mode of the Commandline Client

The UNICORE commandline client (UCC) [15] is a core component of UNICORE 6 and offers full access to the functions of a UNICORE 6 Grid. The UCC includes a batch processing mode, where a set of job files is read and jobs are submitted to the available compute resources. This batch mode can be used for high-throughput computation, where the user just has to generate the individual job files. There is no built-in fault handling, so the user has to deal with job failures herself, for example by re-running failed computations. Fault-handling features could however be added to UCC in the future. Resource discovery is performed by looking up target systems that offer the required application. More detailed brokering, for example by operating system or number of processors is not done by UCC. UCC selects resources using a round-robin strategy. A number of jobs are submitted concurrently, and their status is checked using a polling approach. The total number of concurrent jobs, the number of client threads used, and the polling interval used for job status updates can be controlled. This allows some tuning of the batch mode performance and controlling of the load generated on the Grid. UCC seems well suited for simple batch applications that do not require complex brokering or fault-handling strategies. Its scaling behaviour is fairly good due to its simplicity.

3.2 HiLA Java API

HiLA [14] is a Java API to a UNICORE Grid, offering a simple set of abstractions (such as Grid, Site, Task and File) and a familiar programming model. Using HiLA, applications can make use of Grid resources and run remote computations easily. HiLA can be used to develop high-throughput applications, with expected characteristics similar to the UCC batch mode.

3.3 UNICORE 6 / Chemomentum Workflow System

This workflow system has been developed within the European Chemomentum project [12]. It is fully integrated with the UNICORE middleware since the 6.1 release. The workflow system adds two layers to the basic UNICORE 6 architecture. A workflow engine layer deals with execution of high-level workflows, while a service orchestrator deals with resource discovery, selection and job execution. The system has been designed to allow easy scaling. The service orchestrator component is stateless in the sense that it operates on a per-job basis. Thus, multiple service orchestrator instances can be deployed to allow load-balancing. This workflow system can be used as-is for high-throughput computations, because it supports semi-automated data splitting and the service orchestrator component. The system is very simple to use for end-users, and needs no further programming or customisation work. The workflow engine receives a simple XML description of the task to be performed, with some workflow options that control the data splitting. The workflow engine then auto-generates a more detailed workflow with all sub-tasks specified.The sub-tasks are then sent to the service orchestrator which executes them on a suitable UNICORE 6 resource. The workflow system supports fault handling in the sense that failed computations can be repeated, and the system can deal with disappearing and newly appearing execution systems. At the time of writing, more elaborate, rule-based fault-handling is still under some development. The split and merge operations are performed internally, because a suitable UNICORE application for data splitting/merging has to be available on the Grid, and will be invoked automatically by the workflow engine. A special Resource Information Service (GRIS) is used for resource discovery. This service keeps Grid resource information which is periodically updated. Resource selection is performed by a brokering sub-component of the service orchestrator, which queries the GRIS and selects an execution resource based on current GRIS data and a set of configurable strategies. The basic strategy is based on application availability, combined with a round-robin approach in the common case of multiple execution host candidates. The Service orchestrator deals with job control, submitting jobs to the selected resources, checking their status, and sending notifications to the workflow engine when jobs succeed or fail. At the time of writing it is not yet clear how well this architecture scales in practice with increasing number of Grid sites, due to the limited deployment experiences.

4 Tuple Space Based Approach

The bottlenecks when using Grid systems for high-throughput computations are usually the resource discovery and selection processes. These are expensive operations, involve many web service calls, and tend to scale badly with increasing number of Grid resources. To completely bypass this procedures, we propose a different approach based on the tuple space concept.

A tuple space is essentially a shared memory accessible by distributed clients and servers. It stores data as records with typed fields (called tuples). The tuple

space provides a small number of operations to insert, read and remove (take) tuples from the space, using template-based queries. The original concept was designed by David Gelernter and others for the Linda system in the mid-80s [5] and many implementations exist, for example several Java implementations based on SUNs JavaSpaces APIs [7]. Commercial implementations such as GigaSpaces [8] have gotten a lot of publicity recently due to their promise to deliver horizontally scalable, "share-nothing" enterprise architectures.

In an XML centric web-services system such as UNICORE 6 the idea to build a tuple space for XML documents is quite natural. XML-based tuple space implementations are not very common, however. It has been noted that XML and web services might be a promising way forward for Linda-like systems [6] especially in conjunction with web services. A .NET based XML tuple space was implemented by Tolksdorf et al. [9].

In the course of a diploma thesis [10], a tuple space for storing and retrieving arbitrary XML documents has been designed and implemented, based on the WSRFlite web services framework used in UNICORE 6. It is composed of two services, the Space service itself and a WSRF service for storing the tuple space entries. Each entry corresponds to a WS-Resource. Reading and taking entries involves matching the entries in a brute-force manner against a template.

4.1 Job Execution Using the Tuple Space

The XML documents used for realising the job execution application look as follows

```
<Job xmlns="http://www.unicore.eu/unicore6/spaces/job">
   <JobID>
   <ServerJobID>
   <ServerID>
   <Status>
   <Address>
   <JSDL>
</Job>
```

Here, the "JSDL" element stores the job description, and the "Status" field can take the values NEW, SUBMITTED and DONE. The other fields are used to store information relevant to the client, such as the endpoint reference of the UNICORE 6 job management service for managing the job.

As Figure 1 shows, the basic scheme is as follows:

- The client submit jobs to the space
- At the target system, a worker component (“job taker”) takes jobs from the tuple space and submits them to the target system and thus to the underlying UNICORE 6 XNJS execution manager
- When the job is done, the XNJS notifies the worker, and the job is written to the tuple space with status "DONE".
- The client checks for done jobs, and can download results

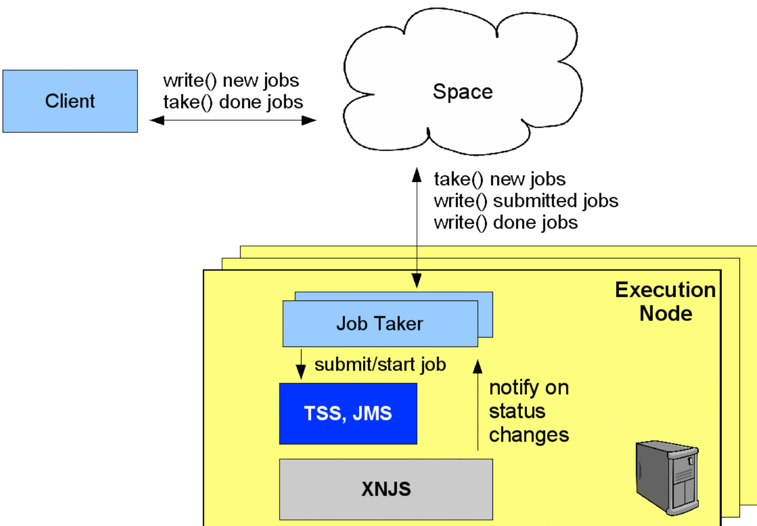


Fig. 1. Tuple space based job execution

This approach promises several advantages to the usual schemes. The tuple space can be seen as a globally shared queue, thus resource usage should be very efficient. The crucial point is that the job takers at the TSS decide when to fetch and submit the next job. This makes applying local scheduling policies trivial. Resource discovery and explicit resource selection by Grid clients are not necessary, nor is publishing of local scheduling information to a Grid scheduler. Another major benefit is that workers can be added (and removed) easily and transparently, without the need to make their presence known to other Grid components such as Grid schedulers or information systems.

The existing implementation is very simple, each job taker will just process exactly one job at a time. Of course, more complex policies are quite easy to implement (for example, on a cluster system it would be better to accept one job per compute node). Another limitation of the current implementation is that there is no well-defined order (e.g. FIFO) in which the jobs will be processed.

The UNICORE Spaces module including a prototype of the job execution application discussed in this section is available on the UNICORE Subversion repository [11]. This prototype does not include any security functionality (except for the standard TLS), the space is freely accessible, and jobs are submitted under the worker node's identity. However, we note that reasonable security mechanisms would be straightforward to add.

5 Some Performance Results

We performed some preliminary performance measurements on a small test Grid composed of six AMD Opteron 2GHz machines, with 2GB of memory, connected

to the LAN with Gigabit Ethernet. They are running SUSE Linux 10.1 and Sun Java 1.5. Each UNICORE 6 service container is configured to use 128Mb of memory, and persistence is activated using the HSQLDB database. Node 1 hosts Gateway, Registry, XUUDB and the Space service, Nodes 2-5 are the workers, and Node 6 is used as client. The worker nodes are configured to run at most two jobs at a time.

Running 100 "Date" jobs (no data staging) on 4 worker nodes using the UCC batch mode took about 120 seconds, where we switched off the download of result files and the checks for application availability.

Our tuple space based job execution performed significantly better. Even using just a single worker, the 100 jobs were finished in about 100 seconds. This shows that in the space-based case there is much less overhead associated with each job.

Using two and four worker nodes, the 100 jobs were finished in 48 and 26 seconds respectively, showing linear scaling in this region. Also, each node consumed an approximately equal share of the total workload.

Similar scaling behaviour can be achieved with higher numbers of jobs, since the client limits the number of concurrent jobs to 100. When this limitation is removed, performance decreases slightly due to the longer lookup times in the space.

We have also measured the average time needed to lookup 100 random entries in the space, while varying the total number of entries. We find the linear increase that would be expected due to the brute-force lookup algorithm. The average lookup times are 12ms for 2000 entries, rising to 40ms for 10000 entries. This indicates a potential bottleneck: as the number of clients increases, the tuple space will become blocked for longer periods of time and the throughput will decrease. More measurements are needed to find the true limits here.

6 Summary and Outlook

The space-based approach presented here is highly promising for applications that do not have complex resource requirements, and that do not requiring high-level Grid features such as co-scheduling. For example, docking or other types of high-throughput screening are very well suited for this approach. Several issues remain though. Most importantly, real-world security requirements still have to be implemented. However, we are convinced that the XML space is flexible enough to handle these requirements. The excellent scalability characteristics would be very hard to achieve with other, more traditional Grid tools. Also, the space-based approach is very simple, and does not require complex broker and scheduler components.

Our preliminary performance numbers confirm the expectations we had when designing the system. Still, it remains to be seen how well the space-based approach scales up to higher numbers of worker nodes and concurrent clients. It should be expected that the performance of the central Space service will degrade under heavier loads, and load-balancing and clustering techniques will have to

be employed. Also, the storage and lookup techniques for tuple space entries will have to be improved, to avoid needless searches and XML matching. For example, in the job execution application it would be highly beneficial to partition the tuple space using the job's Status field, which is used as the major search criterion in the application. Of course, the tuple space itself is generic, so the application programmer needs to provide some hints to the system how to do the partitiong. Similarly, indexes could be built on selected parts of the XML to decrease lookup times.

In summary, the basic overall simplicity of the tuple space concept and the applications in facilitates is clearly attractive and fits in very well with the UNICORE philosophy.

Acknowledgement

Part of this work was funded by the European Commission in the Chemomentum project (IST-5-033437).

References

1. Streit, A., Erwin, D., Lippert, T., Mallmann, D., Menday, R., Rambadt, M., Riedel, M., Romberg, M., Schuller, B., Wieder, P.: In: Grandinetti, L. (ed.) Grid Computing: The New Frontiers of High Performance Processing Advances in Parallel Computing, vol. 14, pp. 357–376. Elsevier, Amsterdam (2005)
2. Initiative for grid-enabled drug discovery against neglected and emergent diseases (April 2008), `http://wisdom.eu-egee.fr`
3. UNICORE website (April 2008), `http://www.unicore.eu`
4. UNICORE 6 overview (April 2008), `http://www.unicore.eu/documentation/files/Unicore6Overview.pdf`
5. Gelernter, D.: Generative communication in Linda. ACM Trans. Program. Lang.Syst. 7, 80–112 (1985)
6. Wells, G.: Back to the future with Linda. In: Second international workshop on coordination and apaptation techniques for software entities (in conjuction with ECOOP 2005), Oslo 2005 (April 2008), `http://wcat05.unex.es/Documents/Wells.pdf`
7. SUN Microsystems: Jini (April 2008), `http://java.sun.com/software/jini`
8. Gigaspaces commercial JavaSpaces implementation (April 2008), `http://www.gigaspaces.com`
9. Tolksdorf, R., Liebsch, F., Nguyen, D.M.: XMLSpaces.NET: An extensible Tuple Space as XML Middleware. In: 2nd International Workshop on.NET Technologies 2004 (submitted) (April 2008), `http://www.ag-nbi.de/research/xmlspaces.net`
10. Schumacher, M.: Realisierung eines XML-Tupelraumes unter Verwendung des Web Service Resource Framework, University of Applied Science Aachen/Juelich (February 2008)
11. UNICORE Spaces Subversion repository (April 2008), `http://unicore.svn.sourceforge.net/svnroot/unicore/contributions/unicore-spaces/trunk`

12. Chemomentum: Grid-Services based Environment for enabling Innovative Research (April 2008), http://www.chemomentum.org
13. Schuller, B., Demuth, B., Mix, H., Rasch, K., Romberg, M., Sild, S., Maran, U., Bała, P., del Grosso, E., Casalegno, M., Piclin, N., Pintore, M., Sudholt, W., Baldridge, K.K.: Chemomentum - UNICORE 6 based infrastructure for complex applications in science and technology. In: Bougé, L., Forsell, M., Träff, J.L., Streit, A., Ziegler, W., Alexander, M., Childs, S. (eds.) Euro-Par Workshops 2007. LNCS, vol. 4854, pp. 82–93. Springer, Heidelberg (2008)
14. HiLA High-level API for Grids (April 2008), http://www.unicore.eu/community/development/hila-reference.pdf
15. UNICORE commandline client (April 2008), http://www.unicore.eu/documentation/unicore6/manuals/ucc

The Chemomentum Data Services – A Flexible Solution for Data Handling in UNICORE

Katharina Rasch[1], Robert Schöne[1], Vitaliy Ostropytskyy[2], Hartmut Mix[1], and Mathilde Romberg[3]

[1] Technische Universität Dresden, Dresden, Germany
[2] University of Ulster, Coleraine, Northern Ireland
[3] Forschungszentrum Jülich GmbH, Jülich, Germany

Abstract. This paper introduces the Chemomentum data services, a UNICORE-based flexible solution for managing large amounts of data and metadata produced in a Grid. In order to store and manage the increasing amounts of data produced in Grid environments, a highly scalable and distributed Grid storage system is needed. However, the simple storage of data is not enough. To allow a comfortable browsing and retrieving of the data, it is crucial that files are indexed and augmented with metadata. This paper analyses integrated solutions that already provide these functionalities for their features and shortcomings. Incorporating the conclusions drawn from the examination, an architecture for a revised data management solution is presented. This system provides the means to store files with augmenting extensible metadata. It allows also to browse through data using metadata, handle ontologies and transparently access external data sources. In the current stage, most of these functionalities are implemented and running in a distributed environment.

1 Introduction

Grid technologies are starting to realise their large potential to provide innovative infrastructures for complex scientific and industrial applications. The UNICORE 6 Grid computing solution [1] already provides a seamless, secure, and intuitive access to distributed Grid resources. However, to make Grids more useful for knowledge-oriented applications such as decision support and risk assessment, more effort in the fields of semantics, metadata and knowledge management in distributed, heterogeneous environments is needed.

The Chemomentum project [2] aims to fill these gaps by taking up and enhancing state-of-the-art Grid technologies and applying them to real-world challenges in computational chemistry and related application areas. The Chemomentum project specifically supports the European REACH (Registration and evaluation of chemicals) [3] initiative aimed at optimising risk assessment strategies.

To be in line with the REACH initiative it is crucial that all results of calculations can be subsequently evaluated and reproduced. This means that not only final experiment results may need to be stored permanently, but also intermediate results and metadata that describe the provenance of the result data,

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 84–93, 2009.
© Springer-Verlag Berlin Heidelberg 2009

e.g. the workflow and applications that produced the data and information on the user who ran the workflow.

It is foreseeable that a considerable amount of data and metadata will be produced. Depending on the type of executed workflow and the level of detail, the size of metadata can reach hundreds of kilobytes, the data even up to gigabytes. Therefore a highly scalable, distributed and decentralised Grid storage solution is needed. The storage of data and augmenting metadata is the fundamental requirement for such a storage system. An important feature is the possibility to define sets of metadata, which describe data used or generated within specific workflows. Clearly defined and well structured metadata sets are crucial to provide adapted user interfaces for diverse purposes.

A transparent access to external data sources (e.g. web-based chemical databases) is also desirable. Data from such sources could then be included as input data of a workflow just like data in the internal storage. Ontologies describing synonyms in a specific context can be used to ease the retrieving of metadata. If a user searches for data connected with the chemical name 'H_2O' additional results for 'water' or 'dihydrogen monoxide' could be returned as well. By providing an integrated conversion of values between scientific units, the advantages of using ontologies can be even further extended.

As one solution, the data management system developed in the Chemomentum project is presented in this paper. The system is not limited to the scientific domains in focus of the Chemomentum project, but is a general approach to the challenges of a Grid storage solution.

2 Related Work

Handling of data and metadata in a Grid environment has been described before. Most of these approaches, however, provide only data access but no or insufficient handling of metadata. Nevertheless, there are existing solutions for the Globus[4] and gLite[5] Grid middlewares.

The Storage Resource Broker SRB [6] handles data and metadata in a data Grid, a digital library, a persistent archive, or a distributed file system. It has been developed by the San Diego Supercomputer Center and is commercialised by Nirvana. The SRB provides a uniform data access to different storage types over a network as well as the replication of files. It is often used in Globus Grid projects. The SRB could not be used because the support for metadata extension and schema handling is not sufficient. The need for a commercial license to deploy SRB within Chemomentum was an important disadvantage.

Another solution that offers both, data and metadata handling, is the Arda Metadata Catalogue Project AMGA [7] which is used with the gLite Grid middleware. It supports user-defined metadata to describe the data stored in the system. These schemas are, however, not shared between users, a crucial point against the adoption for the Chemomentum project. The AMGA server is a monolithic C++ implementation, making it platform dependent and inhibiting the installation of just a subset of the functionalities.

A data and metadata management system that also integrates ontologies, the conversion of scientific units and the transparent access to external data sources could not be found.

3 Overview of the Chemomentum Data Management System

The data management system within Chemomentum provides data storage and retrieval functionality and a global data view independent of the actual data location. A crucial point in designing the system was to build lightweight, specialised services dealing with the different types and sources of data and metadata. The services have well-defined interfaces that allow the installation and running of single services or of subsets of the complete system. The interfaces also support the easy plugging in of extensions, e.g. to access other storage systems.

The heart of the data management system is the Documented Data Space DDS (see figure 1). It is composed of metadata databases, data storages and a location database. The data storages contain data in flat files, typically input and output data produced by workflows. The location database acts as a global file location directory by indexing those files and assigning them globally unique logical names. The metadata databases contain metadata that describe the files in the data storages, referencing them by their logical names. Access to the components of the DDS is provided by a set of three specialised services.

The central interface to the data management system is the Data Management System Access Service (DMSAS). It forwards service requests to the appropriate service(s), collects the results and returns them to the requester. The Database

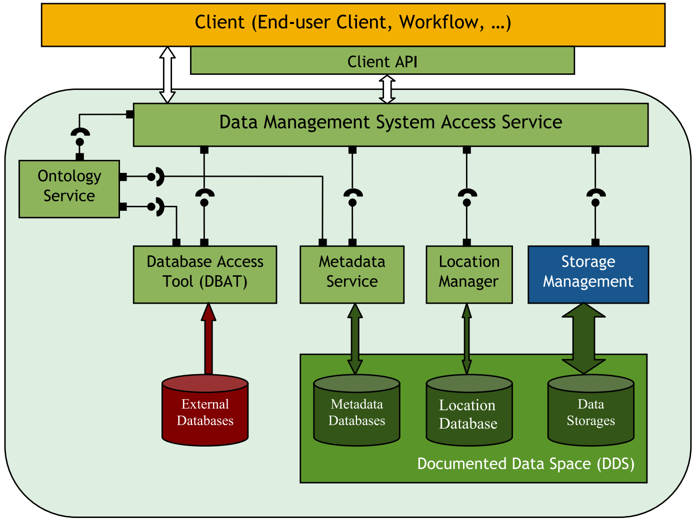


Fig. 1. System architecture

Access Tool (DBAT) serves as a uniform interface to external data sources. It transforms a query to the data management system into the native query language used by the external data source, queries the source and transforms the result back into a format the client understands. The ontology service provides the access to external ontology services and knowledge about types, units and vocabulary to interpret the data.

For the other side of the communication, which includes the workflow system, GUIs and other clients, a special API is provided. The ClientAPI procures objects and methods which enable programmers to access the data management system without any knowledge of the underlying web service schemas and security issues. Built on top of it, special Eclipse RCP [8] views are implemented, which enable the users to create new metadata schemas and to create, query and download data and metadata.

The data management system uses the wsrflite web service framework of the UNICORE 6 Grid middleware. UNICORE 6 implements the Grid standard WSRF, compliant with the Open Grid Forum's Open Grid Services Architecture (OGSA)[9]. It offers strong security based on industry standards such as the X.509 public key infrastructure. The communication with UNICORE 6 services is protected by mutual authentication. All services that form the data management system are implemented as web services that can be run distributed across the Grid.

4 Detailed Architecture

Following the short introduction of the Chemomentum data management solution, this section will now focus in detail on the key characteristics of the system.

4.1 Metadata Modelling

While Chemomentum itself is specifically aimed at the computational chemistry domain, the ambition is to develop Grid technologies applicable in any scientific, economic or other domain. Hence, the ability to cater for arbitrary, extensible metadata schemas instead of limiting the available metadata items to a fixed set was a crucial design criteria for the data management system. The naive approach of using a general metadata model like RDF (Resource Description Format)[10] for this was deemed unfeasible. Having a completely unrestricted modelling of metadata can be more limiting than a fixed set of items. The user needs some knowledge about the data's semantics to interpret the results. Without a restricting metadata schema describing the semantics, a set of metadata triples is just a bunch of information, difficult to interpret and process.

Instead, a metadata model similar to a relational database was chosen for Chemomentum. The metadata schema for a scientific domain is declared (and can also be extended) by a scientific administrator. The schema includes the basic information about names and data types of metadata items, but also further

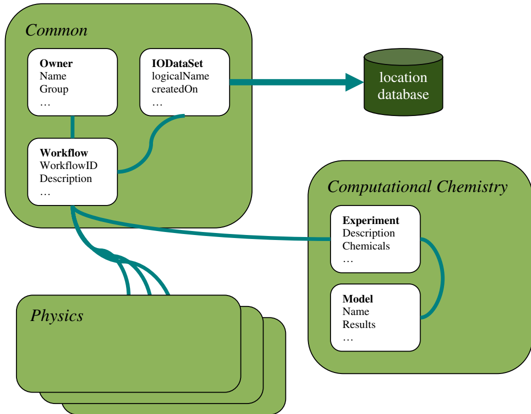


Fig. 2. Metadata modelling

information like a detailed description and units. Attributes can be linked to each other and to logical names of files.

Figure 2 shows an exemplary metadata schema setup. The domain 'common' is a fixed domain that contains basic provenance information, e.g. the workflow that produced the data, the owner of the data and the applications used. The other domains are defined additionally. To gain knowledge about a domain, a user client can request the domain's metadata schema from the data management system and provide the user with a dynamically created graphical interface adapted to the domain.

Aside from manual uploads, most of the metadata is automatically filled by the UNICORE 6 Workflow System that runs the applications and uploads the resulting files into the DDS. While this is a straightforward process for the fixed 'common' domain, the handling of additional domains needs further input from the scientific administrator. The Workflow System has to be provided with the knowledge of how to extract metadata from the input and output of applications and how to map it to the metadata items required by the schema.

Users can manually annotate metadata with additional information like rating and comments for files. Also, metadata which could not be retrieved automatically, can be supplemented later.

4.2 The Documented Data Space

The Documented Data Space (DDS) is split into three types of data storages: the metadata databases, the location database and the file storages. The location database acts as a global file catalogue that maps globally unique logical file names onto the actual physical locations of files and directories. The files are only referenced by their logical names in the system. This mechanism allows for an easy migration of files from one file storage to another. The location database also supports the replication of files, increasing the reliability as well as the performance

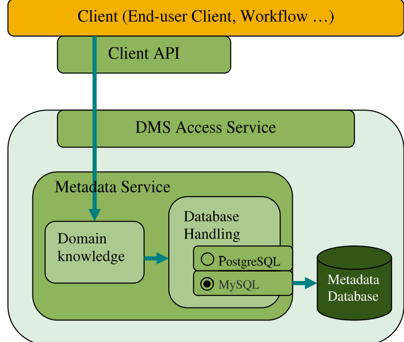


Fig. 3. Database handlers

of the system. Access to the location database is given through the Location Manager. This service provides the functionalities to store, look-up, update and delete locations.

The data files themselves are stored on file storages. The file storages are implemented using the UNICORE 6 Storage Management Services that offer access to storages in a Grid. The current transfer is based on OGSA ByteIO standard, but can be extended to other protocols as well.

Metadata that augment the files in the file storages is kept in the metadata databases of the DDS, as described in section 4.1. The metadata databases are accessed by the Metadata Service, which offers functionalities to store, query, update and delete metadata. The metadata modelling acts as an additional layer above the underlying database systems (see figure 3). All metadata exchanged between the client and the Metadata Service operates on one or more domain schemas and not directly on the database system. Pluggable database handlers provide the mapping between the client data and the specific underlying database technology. Special care is taken for write access to metadata items marked as provenance. The DDS implements a row-based security on the data in the metadata databases. Access control lists mark whether a user, group or virtual organisation has the right to read the data, to write it or to change the permissions. Users with high-level security and privacy needs can also provide their own private DDS on their own site.

4.3 The Database Access Tool

The Database Access Tool (DBAT) provides a comfortable access to external data sources. External data sources can be various database systems, web services or simply flat files. Like the Metadata Service, the DBAT is called only by the Access Service. The DBAT extracts the name of the external data source to be accessed from the query and transforms the query accordingly to the native query standard of the data source. The native query standard is, for example, SQL for data in relational database systems and XML-based service calls for

data provided by web services. The transformation is done by using the aforementioned database handlers. The result set of the query is returned to the Access Service for further processing.

Just like the Metadata Service, the DBAT uses metadata schemas to know the correct set up for the scientific domains it supports. In the current status of the project the Ecotoxicology[11] databases Aquire and Terretox as well as the Protein Data Bank (PDB) [12] are implemented.

4.4 The Ontology Service

The Ontology Service supports queries by providing additional, domain specific information about data - for example, synonyms to broaden queries to data services or knowledge about the conversion of values between different units.

To broaden a query, the Ontology Service contacts external ontology services to examine the query for data items that can be represented in a different fashion. At the present stage, for example, the external ontology service ChEBI[13] is used to look up the molecular names of small molecules. The synonyms of data items are then aggregated into the query. When the broadened query is executed, it will return more exhaustive results then the original query.

The Ontology Service also provides the means to automatically convert values in requests between user-provided units and the units used in the databases. The set of units supported by the Ontology Service is not fixed, scientific administrators can set up own unit groups and conversions.

4.5 The Data Management System Access Service

The Data Management System Access Service (or more conveniently Access Service) is the sole entry point into the data management system. Therefore, only one open port is needed to operate the system. The Access Service is a lightweight service that can easily be run in several parallel instances to avoid bottlenecks. It implements the logic to bundle the functionalities of the underlying services to a higher-level interface, that offers comfortable methods to query, store, update and delete data and metadata.

The user is presented with a uniform interface for queries to the DDS and to external databases. Queries made are directed to the appropriate services. The Access Service also manages distributed queries to multiple Metadata Services. With the help of the Ontology Service, queries are automatically broadened to improve the results and any unit conversions necessary between the user's units and the units used in the system are performed. Files requested by the users are looked up in the DDS and made available for an easy download.

In particular the process of storing data and metadata demands the interplay of multiple services. The Access Service coordinates the following tasks:

1. Check the metadata for validity.
2. Perform necessary unit conversions.
3. Upload the file(s) to the storage(s).

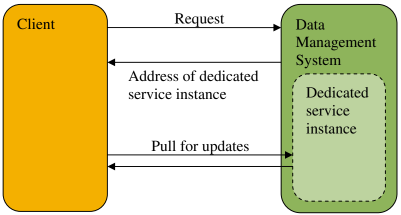


Fig. 4. Asynchronous processing

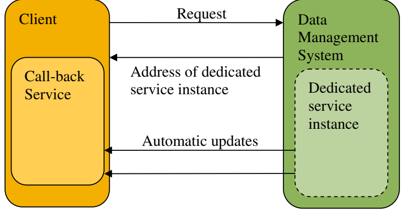


Fig. 5. Automatic call-backs

4. Register the files with the Location Manager.
5. Store the metadata in the metadata database.

A transaction mechanism automatically rolls back all changes already made if one of those steps fails.

Considering the huge amount of data that can be processed in one request, the data management system uses asynchronous execution (see figure 4). The Access Service spawns a dedicated service that handles the request and returns its address to the user. The user can then poll this service for the current status of the processing. Alternatively, the user can provide the address of a call-back service that is updated automatically by the system (figure 5).

4.6 The Client API and the Eclipse Plugin

The Client API provides users and developers with a convenient Java based interface to the data management system. Its key features are the comfortable support for handling large sets of metadata items and for the asynchronous execution mode.

The Client API supports the access to the system in various levels of complexity. It consists of low-level implementations that resemble the web service

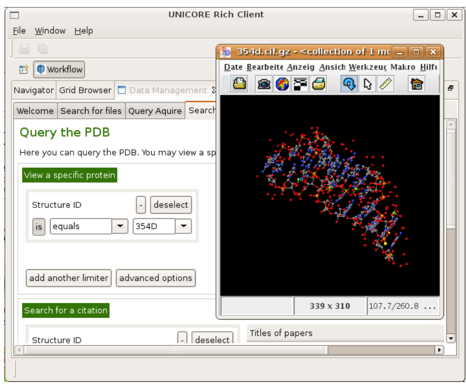


Fig. 6. Query external data sources (e.g. PDB)

interface, but it includes also high-level implementations that allow the client developer to start a request with a single line of code.

On top of the Client API an Eclipse Plugin has been developed, which smoothly integrates with the UNICORE RCP client. It can be used to access the data management system and, for example, upload and download files, browse through metadata or access the external databases (figure 6).

5 Current Status and Future Work

In the current stage of development, all services described here are implemented and can be installed on any server that is running a Java Virtual Machine in version 1.5 or higher. The components have already been installed in a distributed fashion and are running as part of a testbed for the Chemomentum project. GUI components that allow browsing and storing data and metadata are available and are in use for testing the system.

Within the year 2008, the final version of the Chemomentum software – including the data management system – will be available. In addition to the functionality presented in this paper, this version will support more external databases and contain a sophisticated administration interface. Additional GUI components for a simple and flexible access to the system will also be provided.

6 Conclusions

The Chemomentum data management system is a flexible, distributed and user-friendly approach to data management in the Grid. Because of its extensible metadata system, it is not limited to specific use cases, but can be used in arbitrary scientific, economic or other fields.

The data management system can be deployed on any server running the Java Virtual Machine. Its modular design allows also for a distributed deployment on multiple servers and sites. The data in the data management system is protected by sophisticated security and safety solutions based on the UNICORE 6 framework. The data management system supports web service standards like WSRF and OGSA ByteIO. The Eclipse Client and the ClientAPI allow an easy integration into other projects without knowledge of the internal handling of data and meta data. Despite of its UNICORE background, the system is deployable also in non UNICORE based Grid environments. This makes it an excellent solution for Grid data management.

Acknowledgement

This work has been funded by the European Commission under contract no. IST-5-033437.

References

1. Riedel, M., et al.: Web Services Interfaces and Open Standards Integration into the European UNICORE 6 Grid Middleware. In: Proceedings of 2007 Middleware for Web Services (MWS 2007) Workshop at 11th International IEEE EDOC Conference The Enterprise Computing Conference (2007)
2. Schuller, B., Demuth, B., Mix, H., Rasch, K., Romberg, M., Sild, S., Maran, U., Bała, P., del Grosso, E., Casalegno, M., Piclin, N., Pintore, M., Sudholt, W., Baldridge, K.K.: Chemomentum - UNICORE 6 based infrastructure for complex applications in science and technology. In: Bougé, L., Forsell, M., Träff, J.L., Streit, A., Ziegler, W., Alexander, M., Childs, S. (eds.) Euro-Par Workshops 2007. LNCS, vol. 4854, pp. 82–93. Springer, Heidelberg (2008)
3. REACH (Registration and evaluation of chemicals), Regulation (EC) No 1907/2006 of the European Parliament and the Council, http://echa.europa.eu/reach_en.html
4. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. In: Jin, H., Reed, D., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 2–13. Springer, Heidelberg (2005)
5. Laure, E., et al.: Programming the Grid with gLite. Computational Methods in Science and Technology 12(1), 33–45 (2006)
6. Rajasekar, A., et al.: Storage Resource Broker - Managing Distributed Data in a Grid. Computer Society of India Journal, Special Issue on SAN 33(4), 42–54 (2003)
7. Santos, N., et al.: Distributed Metadata with the AMGA Metadata Catalog. In: Workshop on Next-Generation Distributed Data Management HPDC-15, Paris, France (2006)
8. Eclipse Rich Client Platform homepage, http://wiki.eclipse.org/index.php/Rich_Client_Platform
9. Foster, I., et al.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In: Open Grid Service Infrastructure WG, Global Grid Forum (2002)
10. McBride, B., et al.: The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. In: International Handbooks on Information Systems, Handbook on Ontologies (2004)
11. Ecotoxicology databases homepage, http://cfpub.epa.gov/ecotox/
12. Berman, H.M., et al.: The Protein Data Bank. Nucleic Acids Research 28, 235–242 (2000)
13. Degtyarenko, K., et al.: ChEBI: a database and ontology for chemical entities of biological interest. Nucleic Acids Research 36, 344–350 (2008)

A Reliable and Fast Data Transfer for Grid Systems Using a Dynamic Firewall Configuration

T. Oistrez, E. Grünter, M. Meier, and R. Niederberger

Research Centre Juelich, Juelich, Germany
{t.oistrez,e.gruenter,m.meier,r.niederberger}@fz-juelich.de
http://www.fz-juelich.de/jsc

Abstract. Firewalls separate areas of different security requirements. This major task leads to problems regarding the network connectivity and performance of various applications. In particular within distributed systems, like a Grid an unobstructed communication, which is essential for using distributed resources is not possible. Furthermore Grid applications often use multiple ports dynamically and in parallel. This raises the challenge of a dynamic configuration of firewalls. This paper shows a solution based on UDP hole punching and describes the implementation of a UNICORE transfer service using this technology to perform direct high speed file transfers.

1 Firewalls and Filtering of Network Traffic

Firewalls are used to separate areas of different security policies from each other. Their major task is to protect computing resources against unauthorized access and misuse. In order to achieve this task the firewall system processes certain information which is used as a basis to decide if a message may pass through the firewall. The firewall administrator defines a ruleset which represents the implementation of the local security policy. The network traffic is divided into classes of those packets that will be forwarded to the destination and those which will be rejected. This ruleset is a baseline to different tests that will be applied to each incoming packet. The firewall checks IP addresses and ports of the appropriate protocol headers. In addition, stateful packet inspection engines use connection states.

Firewalls store state information primarily when a TCP stream is discovered because TCP is a connection oriented protocol. Connections may be in one of four different states: half open, established, half closed and closed state. Therefore it is reasonable to differentiate which state a connection has entered. Additionally, TCP is a reliable protocol, i.e. if any TCP segment is lost during transfer a retransmission is triggered due to missing acknowledgments discovered at the sender. Further information can be found in [1].

The User Datagram Protocol (UDP) is a non-reliable and non-connection oriented transport layer protocol. An application that uses UDP has to make sure that the data is successfully transmitted. Although no connections exist, firewalls use a simple mechanism to simulate a connection. Figure 1 shows a

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 94–102, 2009.
© Springer-Verlag Berlin Heidelberg 2009

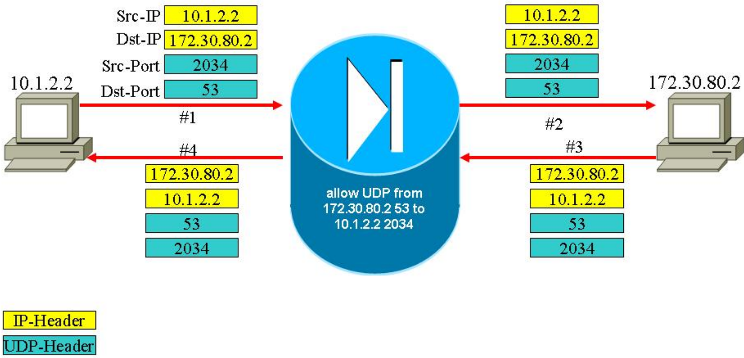


Fig. 1. Firewalls and UDP

client behind a firewall being allowed to send UDP packets to outside, which sends a UDP datagram to a DNS server outside the companies network.

The client generates the UDP datagram and sends it to the firewall (Figure 1 #1). The firewall examines the datagram and forwards it to the destination (#2). According to the information that has been gathered the firewall adds an entry to its connection table including source IP address, source port, destination IP address, and destination port. However, this results in a dynamically generated access rule like

allow UDP from 172.30.80.2 port 53 to 10.1.2.2 port 2034

which is valid for a certain configurable period of time. This rule guarantees that as long as the timeout has not been reached, UDP replies from the server (#3) to the client can traverse the firewall (#4).

2 Grid Applications and Firewalls

A Grid is a distributed system which makes resources like compute power, storage capacity, and distributed data available to its users. It forms an union of geographically distributed, independent organizations sometimes referred to as virtual organization. The usage of available resources takes place statically or dynamically at run-time according to the requirements of the user and/or the application.

Grid applications often need high transfer rates and small latencies. Moreover, these applications transfer large data sets which must arrive reliably and as fast as possible at the destination. One approach to satisfy these needs is the usage of several parallel connections. GridFTP [2] can serve as an example here: it mandates that multiple parallel data connections are always established from the sender to the receiver. Thus, a server running GridFTP must be accessible from outside on a broad port-range. Statically opening these ports at the firewall leads to severe security problems because the open server ports can be used by

malicious software while not in use by a current GridFTP transfer. GridFTP is a fast solution but its insecure design disqualifies it for use in most production networks.

An alternative is the dynamic configuration of firewalls. This should satisfy the following:

1. It can be integrated smoothly into an existing administrative security framework.
2. It can be used in open source and in commercial solutions.
3. It keeps the communication between the partners open for the shortest possible time.

Currently there are different solutions to configure a firewall dynamically. These solutions are either proprietary and vendor specific such as Cisco PIX [3] or they support only certain kinds of firewalls. CODO [4] is such a solution. This document presents a novel approach to dynamic configuration of firewalls. It uses a mechanism comparable to UDP hole punching a commonly used NAT traversal technique.

3 UDP Hole Punching

NAT traversal through UDP hole punching is a method for establishing bidirectional UDP connections between Internet hosts in private networks using NAT [5]. It takes advantage of firewalls that simulate connections for UDP traffic.

Prerequisites to use UDP hole punching are the following:

- the local firewall allows outbound UDP connections
- the local firewall simulates connections for UDP data transfer as described in section 1.
- a relay server exists.

The relay server is a central part of this concept. Each client connects to the relaying server using a persistent TCP connection. Simultaneously the relay server recognizes the IP addresses of the clients. It does not even matter if any client connects to the public network through a NAT device because the public IP address is notified.

The following describes how two clients in different security domains establish a UDP connection using UDP hole punching. The initiator (client A) sends a TCP segment to the relaying server C, see figure 2 (message #1). This contains the information that client A wants to talk to client B using a UDP source port, e.g. 4711. The server notifies client B that client A has the public IP address x.x.x.x and that it expects a UDP connection on port 4711 (#2). Client B sends the preferred UDP port, e.g. 8822 to the relaying server and simultaneously it sends a UDP datagram from source port 8822 to destination port 4711 to client A (#3).

Client B's local firewall forwards the UDP datagram, creates a connection entry and the dynamic access rule which allows responses to traverse the firewall.

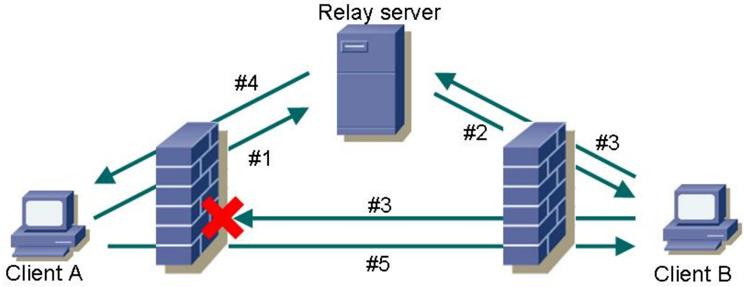


Fig. 2. UDP Hole Punching

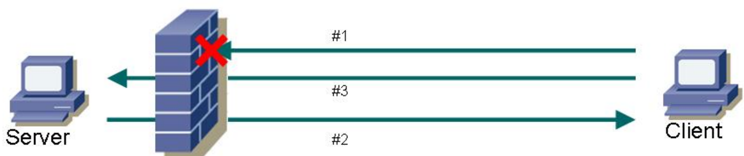


Fig. 3. UDP hole punching with netcat

Client A's local firewall rejects the packet but this does not matter at all. The relaying server C informs client A via the existing TCP connection between A and C that client B is accessible on IP address y.y.y.y and UDP port 8822 (#4).

Client A now sends a UDP datagram from source port 4711 to 8822 (#5). Client A's local firewall now creates the dynamic entries. However, the dynamic entry in B's local firewall is still active and valid, so that the UDP datagram from A to B passes the firewall. Now the communication channel is established although the static ruleset of each firewall would normally deny inbound connections according to the parameters of the protocol headers.

The concept of UDP hole punching can easily be demonstrated using *netcat*. *netcat* is a networking utility which reads and writes data across network connections. It can use TCP/IP and UDP/IP [6]. It is available on most Linux systems. The server resides behind a firewall and listens on port 4711:

server# netcat -u -l -p 4711

A client from the outside tries to connect to the UDP port 4711 on this server behind the firewall, see figure 3(#1):

client# echo Hello | netcat -p 8822 -u server 4711

This UDP connection is not allowed by the local firewall. The UDP datagram is dropped and nothing happens. Now the server sends a UDP datagramm to the client outside the firewall and punches a hole into the firewall (#2). The local firewalls allows outbound UDP communication and forwards the packet

towards the destination. Because of the algorithm used to simulate a connection the firewall passes any reply of the client to the server:

server# echo Hello | netcat -p 4711 -u client 8822

After this initial exchange the datagram from client to server is allowed to pass the firewall (#3):

client# echo Hello | netcat -p 8822 -u server 4711
server# netcat -u -l -p 4711
Hello

This simple example works on any linux system and with different firewalls. We successfully tested it with iptables [7] and Cisco PIX [3].

Because TCP uses connection states that are examined by firewalls, it is no alternative here. It is not possible to make both firewalls believe that a connection has been opened from inside.

4 A Transfer Service Using UDP Hole Punching

For a file transfer based on UDP Hole Punching, UNICORE [8] serves as a perfect environment. Many concepts described in section 3 are already realized. There is a central Gateway for all control messages which are encrypted via X.509 certificates. Users and devices are centrally authenticated and authorized once. The existing file transfers for UNICORE are all based on predefined classes which implement basic features and error handling. A new transfer service is derived from these classes and extends them with the specific methods and members for the used technique. Thus, the implementation of a file transfer using the UDP Hole Punching is done by providing the methods to exchange connection parameters like IP address and UDP port and to send the Hole Punching packet and the data using UDP.

A new challenge arises from encrypted UDP data traffic. One approach to encrypt the data could be a simple symmetric algorithm. Because the TCP connections between clients and the gateway are secured via X.509 certificates the exchange of a shared secret could be done via this channel. Moreover it would be useful to declare the encryption of the data transfer as an optional feature. Because data is not always confidential this could speed up the transfer.

The second challenge results from specific requirements of Grid applications. Data transfers should be fast and reliable but UDP is unreliable. Each UDP datagram is an instance of its own and the application has to make sure that all the data has arrived. In fact this means that Grid applications using UDP need an application layer protocol to implement reliability.

This can be accomplished by *UDP-based Data Transfer Protocol* (UDT). UDT uses UDP as transport protocol but it guarantees reliability through an upper layer. The following section describes UDT [9] in general.

4.1 UDP-Based Data Transfer Protocol (UDT)

Like TCP, UDP is a transport layer protocol. Besides the data payload UDP packets only consist of a minimal header containing information about source and destination port, packet length and a checksum. In contrast to TCP, a UDP header contains neither *flags* or *control bits* nor any sequence or acknowledgment numbers. For that reason the protocol itself is not able to read a connection state from a packet. Additionally, it cannot recognize or interpret packet loss. Therefore TCP features relating to a reliable and fair protocol such as establishing or terminating a connection, buffering packets for a resend after loss and avoiding congestion have to be implemented at higher levels.

UDT is such an implementation. UDT is not a protocol of the transport layer like TCP or UDP. It utilizes UDP as transport protocol and provides reliable communication and congestion control on the application layer, thus completely in the user space.

UDT is available as open source. It is designed and implemented by the National Center for Data Mining at the University of Illinois at Chicago. A first internet draft has been released in August 2004 [10]. The latest stable release including documentation can be downloaded from Sourceforge [9].

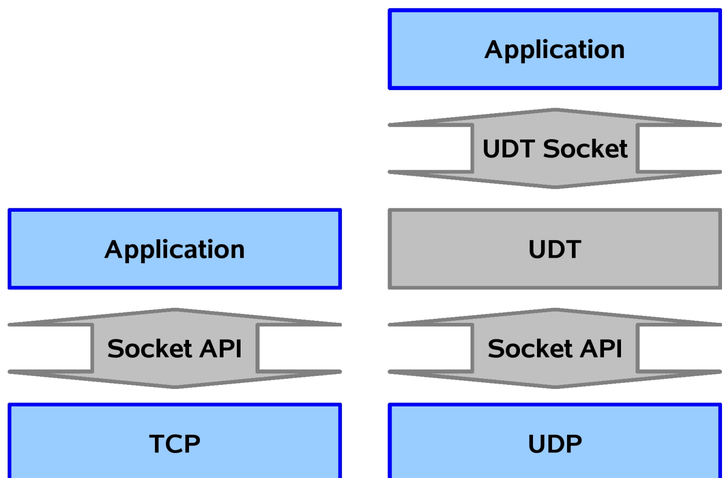


Fig. 4. TCP and UDT sockets

The UDT specific implementation for reliability and congestion control is realised as follows:

- Reliability is achieved by sequencing and acknowledgment. Each UDT packet is assigned a unique increasing sequence number. The receiver will send back acknowledgments and loss reports according to packet arrival. So lost packets will be retransmitted.

– Congestion control: unlike TCP the approach is not window but rate based, meaning that the algorithm does not open up the sender's congestion window, in fact it reduces the inter-packet delay of sent out packets, thus increasing its sending rate. Congestion avoidance uses a special case of the AIMD (Additive Increase Multiplicative Decrease) algorithm; it reduces the increase when getting close to the estimated link bandwidth.

Besides its own congestion control algorithm UDT can also utilize external or custom congestion control algorithms like *TCP Reno* or *TCP BIC* congestion control. That enables developers to find a good agreement between fairness and transfer rate, making the use of multiple parallel connections obsolete in most scenarios.

From a programmer's point of view UDT provides a *C++ API* with a semantic analogue to the TCP sockets (see figure 4).

For further examples, a tutorial and a full list of all UDT functions and references please read the UDT manual [9].

5 The Overall Design

After the concept of UDP hole punching has been described along with extensions for the usage in Grid environments in the previous sections, the overall design of an implementation as a new UNICORE transfer service can be described now.

The implementation is based on the two basic UNICORE classes "FileTransferClient" and "FileTransfer" which are extended by only one WebService method called "initUDT". After transfer objects are created by UNICORE on client- and server-side the process of transfering the file mainly consists of the following steps. The Client immediately starts preparing the UDT connection by binding its UDT socket to a randomly chosen port. It then passes its IP and port number to the server as parameters to the WebService method. The server prepares its own UDT socket and performs the UDP Hole Punching by sending an empty UDP packet to the IP and port that it just got from the client. Then it sends its own connection parameters (IP and port) back as the return value of the WebService method. The Client now can use its prepared UDT socket to connect to the server which is waiting for the connection with a UDT server socket.

During the transfer, both sides collect basic status information like the number of transfered bytes and the time needed. Status information and errors are handled accordingly to the implementation of the standard transfer classes and therefor are easily evaluated.

The fact that UNICORE and its WebServices are implemented in JAVA but UDT is implemented in C++ leads to a hybrid solution using the Java Native Interface (JNI) [11]. JNI is used by JAVA applications to involve functions that are not implemented in JAVA but in a native language. JNI can also be used to load dynamic native libraries and to involve their functions. These functions are declared but not implemented in the JAVA code. The native library is loaded

on runtime. The native functions need to follow some rules concerning the way, parameters are passed. They get pointers to standard JNI functions that give access to JAVA objects. Therefore wrapper functions are necessary to built a bridge between the JAVA classes and the UDT interface. For a more simple design and to minimize the parameter transfer between JAVA and C++, these wrapper functions also contain the code to bind the sockets and to do the Hole Punching.

When using the UDT based transfer, the native library containing the wrapper functions as well as the UDT library need to be available for the underlying hardware architecture and operating system. To simplify the installation, all native code, including the wrapper functions and the UDT source code, is compiled into one single library. The source code as well as a built script for unix like systems are provided as part of the file transfer implementation.

This transfer can be transparently used as an alternative to the commonly used ByteIO mode. Its is working in production networks and has several advantages over GridFTP and ByteIO. On one hand, it is secure enough to be used in most productive networks and on the other hand, it is faster than ByteIO and even than GridFTP. Some performance tests where done in the German X-WIN net with two machines in Juelich and Hannover, connected via 1 GBit/s ethernet. The following table shows the average throughput.

GridFTP (1 Stream)	GridFTP (4 Streams)	UDT 3	UDT 4	ByteIO
81 MBit/s	294 MBit/s	700 MBit/s	930 MBit/s	0.4 MBit/s

These values show how aggressive the standard UDT congestion control works. It must be changed to a fair algorithm or as an alternative, the throughput can be limited by quality of service rules at the networks routers.

6 Summary

Firewalls are absolute essential devices to improve the security of an organization. Consequently, this leads to restrictions regarding network connectivity and performance. Grid applications are affected by firewalls because they need high performance and low latencies. More often they use multiple connections in parallel to speed up the data transfer. To match this requirement of Grid applications static port ranges are opened on firewalls what leads potentially to unauthorized accesses to sensitive resources. Dynamic configuration can ease this problem.

This paper introduced a solution which configures a firewall dynamically based on UDP hole punching to securely establish direct transfers between hosts. The concept has been extended and adapted to the needs of Grid environments and the implementation for UNICORE has been described.

These concepts of Grid UDP hole punching can be seen as a further step in providing solutions for Grid applications dealing with existing firewalls. It

can be easily used by most of the Grid applications known today to overcome time delays until "real" dynamically configurable firewalls are available on the market.

References

1. Richard Stevens, W.: TCP/IP Illustrated I. The Protocols. Addison Wesley, Reading (1994)
2. GT4.0 GridFTP, Globus Toolkit website (August 2006), http://www.globus.org/toolkit/docs/4.0/data/gridftp
3. Cisco Security Appliance Command Line Configuration Guide - For the Cisco ASA 5500 Series and Cisco PIX 500 Series Software Version 7.2, http://www.cisco.com/en/US/docs/security/asa/asa72/configuration/guide/asacfg72.pdf
4. Son, S., Allcock, B., Livny, M.: CODO: Firewall Traversal by Cooperative On-Demand Opening. In: 14th IEEE Symposium on High Performance Distributed Computing (HPDC14), Research Triangle Park (July 2005), http://www.cs.wisc.edu/~sschang/papers/CODO-hpdc.pdf
5. Schmidt, J.: Der Lochtrick - Wie Skype & Co. Firewalls umgehen. In: CT 2006, Heft 17, p. 142. Heise Verlag (2006)
6. The GNU Netcat project (August 2006), http://netcat.sourceforge.net/
7. The netfilter.org project firewall, NAT, and packet mangling for linux (1999 - 2007), http://www.netfilter.org/
8. UNICORE Grid computing Technology UNiform Interface to COmputing REsources (August 2006), http://www.unicore.eu/
9. Gu, Y.: UDT: UDP-based data transfer library - Version 3 (May 2006), http://www.cs.uic.edu/~ygu1/
10. Gu, Y., Grossmann, R.L.: UDT: A transport protocol for data intesive applications Internet Draft, draft-gg-udt-01.txt University of Illinois at Chicago (August 2004)
11. Liang, S.: The Java Native Interface: Programmer's Guide and Specification. Addison-Wesley, Longman, Amsterdam (1999)

Workflow Service Extensions for UNICORE 6 – Utilising a Standard WS-BPEL Engine for Grid Service Orchestration

S. Gudenkauf[1], W. Hasselbring[1], A. Höing[2], G. Scherp[1], and O. Kao[2]

[1] OFFIS Institute for Information Technology, R&D-Division Business Information Management, Escherweg 2, 26121 Oldenburg, Germany
{stefan.gudenkauf,guido.scherp}@offis.de,
hasselbring@informatik.uni-oldenburg.de

[2] Technische Universität Berlin, Faculty IV - Electrical Engineering and Computer Science, Dept. of Telecommunication Systems, Complex and Distributed IT Systems, Einsteinufer 17, 10587 Berlin, Germany
{andre.hoeing,odej.kao}@tu-berlin.de

Abstract. The BIS-Grid project[1], a BMBF-funded project in the context of the German D-Grid initiative, focusses on realising Enterprise Application Integration using Grid technologies to proof that Grid technologies are feasible for information systems integration. Small and medium enterprises shall be enabled to integrate heterogeneous business information systems and to use external Grid resources and services with affordable effort.

In this paper, we describe service extensions to UNICORE 6 to use an arbitrary WS-BPEL workflow engine and standard WS-BPEL to orchestrate stateful, WSRF-based Web Services, also called Grid Services. Thereby, we focus on how to combine the arbitrary workflow engine with UNICORE 6, and on how to access workflows and workflow instances. The workflows itself are also provided as Grid Services, realised by a Workflow Management Service that deploys Workflow Services within UNICORE 6, each wrapping a WS-BPEL workflow that is deployed in the arbitrary workflow engine.

1 Introduction

In order to map business processes to the technical system level the integration of heterogeneous information systems - referred to as Enterprise Application Integration (EAI) - is crucial. Thereby, integration is often achieved by service orchestration in service-oriented architectures (SOA). A means commonly used to create SOA are Web Services since they enable service orchestration and hide the underlying technical infrastructure. Modern Grid middlewares such as

[1] This work is supported by the German Federal Ministry of Education and Research (BMBF) under grant No. 01IG07005 as part of the D-Grid initiative.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 103–112, 2009.
© Springer-Verlag Berlin Heidelberg 2009

UNICORE 6[2] and Globus Toolkit 4[3] are based on the Web Service Resource Framework (WSRF) [1], a standard that extends classical, stateless Web Services to be stateful. Such WSRF-based Web Services, also called Grid Services, provide a basis to build SOAs using Grid technologies.

In BIS-Grid we focus on realising EAI using Grid technologies. One major objective is to proof that Grid technologies are feasible for information systems integration. Small and medium enterprises (SMEs) shall be enabled to integrate heterogeneous business information systems and to use external Grid resources and services with affordable effort. To do so, we develop a workflow engine, the *BIS-Grid workflow engine*, that is capable to integrate Grid Services. This engine is based upon service extensions to the UNICORE 6 Grid middleware, using an arbitrary WS-BPEL workflow engine and standard WS-BPEL to orchestrate Grid Services. Also, it propagates service orchestrations as Grid Services. The main reason that led us to the decision to use UNICORE 6 is that UNICORE 6 is a pioneer in adopting Grid standards, since the support of standards is essential for us, especially regarding security. The WS-BPEL workflow engine to be used is ActiveBPEL[4] since it exhaustively supports the WS-BPEL standard, and is well accepted in the business domain as well as in the Grid domain. We refrain from extending well-adopted standards and technologies as far as possible to increase sustainability. Instead, we use service extensions to UNICORE 6 to conceal the WS-BPEL engine by wrapping the message exchange between the engine and Grid Services.

This paper presents a snapshot on our ongoing work and is organised as follows. Related work is presented in Section 2. In Section 3, we present the architecture of the BIS-Grid solution in detail, highlighting the service extensions to UNICORE 6. Finally, in Section 4, we conclude the paper and briefly present our future work.

2 Related Work

Orchestrating stateful Grid Services is in the focus of many German and international projects, for example, the German D-Grid projects Text-Grid[5] and InGrid[6], and the European projects A-WARE[7], Chemomentum[8], and EGEE[9]. Orchestrating Grid Services is also in the focus of several papers. Leymann [8] describes the appropriateness of using BPEL4WS as a basis for Grid Service orchestration since it already fulfils many requirements of the WSRF standard. He concludes that a Grid-specific extension of BPEL4WS is more appropriate

[2] http://www.unicore.eu
[3] http://www.globus.org/toolkit/
[4] http://www.activevos.com/community-open-source.php
[5] http://www.textgrid.de/
[6] http://www.ingrid-info.de/
[7] http://www.a-ware-project.eu/
[8] http://www.chemomentum.org
[9] http://www.eu-egee.org/

than creating new Grid-specific standards. The appropriateness of BPEL4WS for Grid Service orchestration is also examined and confirmed in [5], [10], [2], and [6]. In [4], Dörnemann et al. discuss composing Grid Services by using BPEL4WS. They present a solution that is based on extending the BPEL4WS specification. Emmerich et al. [5] describe the evaluation of reliability, performance, and scalability issues of the open source workflow engine ActiveBPEL on executing a complex scientific Grid workflow. As a preparatory work to this paper, we describe the requirements that apply to a Grid-enabled workflow system in [6].

UNICORE itself provides a workflow system extension for an existing UNICORE 6 installation, originating from the Chemomentum project. This system consists of two UNICORE/X service containers: a workflow engine, processing workflows on a logical level, and a service orchestration layers that concerns the invocation of Grid Services. The workflow engine utilises pluggable domain-specific language (DSL) modules and a generic workflow language internally. It also includes a resource brokering component. Both the UNICORE 6 workflow system and the BIS-Grid workflow engine have in common that they are realised as service extensions to the UNICORE/X service container. However, the UNICORE 6 workflow system does not feature the integration nor the exchangeability of an arbitrary WS-BPEL workflow engine and is therefore less sustainable.

The works presented in the preceding paragraphs mainly focus on scientific workflows instead of business workflows relevant to BIS-Grid. Concerning the use of BPEL (BPEL4WS or WS-BPEL), it is primarily relied on extending or adapting the language, thus creating BPEL dialects. The A-WARE project is one project that addresses business workflows in Grid environments. In [3], the project presents the orchestration of UNICORE 6 services with the help of a standard BPEL engine, relying on a Service Bus that supports adapters to submit jobs to UNICORE. However, workflows are not provided as Grid Services.

When regarding business workflows, another important aspect is choreography. Choreography regards the interaction and cooperation of multiple workflows, potentially traversing organisational borders. Choreography is relevant to Grid environments since Grid-based Virtual Organisations (VOs) may require workflow cooperation. The TrustCoM[10] project, for example, provides a trust management framework for the definition and enactment of collaborative business processes within VOs. BIS-Grid instead focusses on orchestration. It regards itself as a a door opener for SMEs, allowing them to integrate their information systems with technologies that are Grid-compatible, also allowing them to integrate external Web and Grid Services. Nevertheless, this does not prohibit to consider choreography in future.

3 UNICORE 6 Service Extensions

Our UNICORE 6 service extensions mainly consists of two WSRF service types, *Workflow Management Service* and *Workflow Service*, and an arbitrary standard

[10] http://www.eu-trustcom.com/

WS-BPEL workflow engine. In our case this is the open source workflow engine ActiveBPEL. Together, the service extensions and the arbitrary WS-BPEL engine represent the *BIS-Grid workflow engine*. The service extensions are realised as Grid Services within UNICORE 6's service container, the UNICORE/X component. For each workflow deployed with the Workflow Management Service one Workflow Service will be created using a hot deployment mechanism without restarting UNICORE/X. These services manage and access ActiveBPEL. As a standard WS-BPEL workflow engine, it typically orchestrates stateless Web Services and supports only basic security mechanisms, e.g. username-based and password-based authentication. Therefore, advanced security concepts must be provided by the service extensions in the UNICORE/X service container. In [7] we illustrate some considerations on security within the BIS-Grid solution.

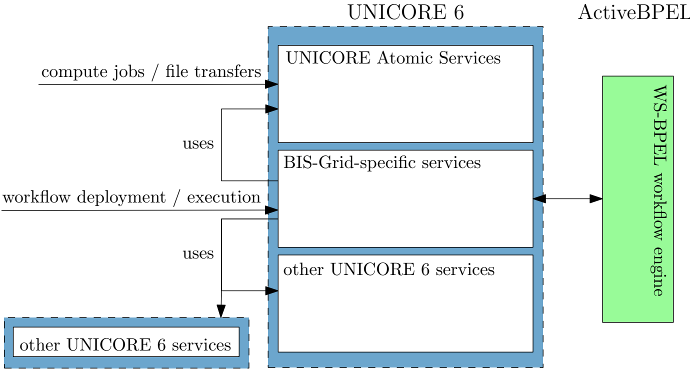
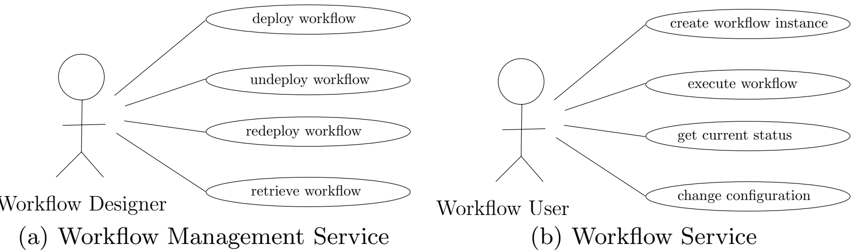


Fig. 1. Overview on the Architecture of the BIS-Grid solution

Figure 1 presents an overview on the architecture of the BIS-Grid workflow engine. Within UNICORE/X, the BIS-Grid service extensions are placed beside so-called *UNICORE Atomic Services* which provide basic functionalities to support Grid computing, and beside other Grid Services that, e.g. may provide access to information systems. One important design decision was to neither extend the WS-BPEL standard nor to modify ActiveBPEL for Grid Service orchestration, although the WS-BPEL 2.0 specification provides an extensibility mechanism that allows to integrate additional functionality without declining the standard. However, the use of proprietary extensions would conclude in a solution that may not be interoperable with future versions of the standard as well as with the engine.

Leaving the WS-BPEL standard and the engine untouched ensures sustainability and flexibility, and allows to exchange the WS-BPEL engine by any other WS-BPEL engine. Figure 1 shows that the ActiveBPEL engine is located behind UNICORE 6. Hence, it can be deployed separately on backend nodes to support load balancing. In [7] we also present our considerations on load balancing the BIS-Grid solution.

Besides all this advantages also some problems arise when using such a decoupled architecture without WS-BPEL extensions. The BPEL code, that is necessary to call a Grid Service is more complex as if we would introduce new grid-specific activities. Even, if a WSRF-resource of a Grid Service is only used for a single call, we have to explicitly create and destroy it using BPEL invoke activities. We tackle this problems by hiding the complexity from the user and the as far as possible from the workflow designer. We plan to extend an existing BPEL editor by introducing new "Grid Activities" that automatically generates the BPEL code. Furthermore, we need some additional BPEL code to solve a mapping problem, that is described in more detail in Section 3.3. The mapping problem will be hidden completely in the Workflow Management Service, that will do some processing steps before the deployment of the BPEL code to the WS-BPEL workflow.



(a) Workflow Management Service (b) Workflow Service

Fig. 2. Use Cases

3.1 Workflow Management

Important workflow management functionalities are workflow deployment, redeployment, undeployment, and retrieval (see Figure 2(a)). These functionalities are realised as a Grid Service, the *Workflow Management Service*, whereas an instance of the service manages exactly one workflow. Workflow Management Service instances are created by a factory that is realised as a standard Web Service within the UNICORE/X service container. This factory offers the following functions:

- `create`: The method creates a new (empty) service instance and returns the corresponding endpoint reference.
- `search`: The method returns a set of endpoint references pointing to service instances that represent workflows accessible to the user. Search patterns can be used to limit the result list.

A Workflow Management Service instance provides the following functions:

- `deploy`: The method deploys a workflow by requiring a *BIS-Grid Workflow Deployment Package*. This package contains a WS-BPEL workflow description and additional resources, e.g. deployment descriptors. Since each Workflow Management Service instance manages only one workflow this method

is blocked after deployment until the `undeploy` method has been called. To modify a deployed workflow the method `redeploy` must be used.

- `undeploy`: This method undeploys a workflow previously deployed with the same Workflow Management Service instance, and destroys the corresponding Workflow Service.
- `redeploy`: This method redeploys a workflow previously deployed with the same Workflow Management Service instance. The effect is regarded the same as if the methods `undeploy` and `deploy` are called consecutively.
- `retrieve`: This method returns the corresponding deployment package of a workflow previously deployed with the same Workflow Management Service instance.

For workflow deployment and undeployment, the Workflow Management Service has to communicate with the WS-BPEL engine and with the UNICORE/X service container to create or remove Workflow Services (see Section 3.2). Therefore, the deployment and undeployment processes can be subdivided into several steps. If one step fails the complete deployment or undeployment process must fail, too, and the preceding steps must be rolled back (if required and possible). The deployment process is as follows:

1. The BIS-Grid Workflow Deployment Package is stored to a previously specified local file space and is unpacked.
2. The deployment package, containing the workflow description and a corresponding deployment descriptor, is checked for correctness and completeness.
3. The WS-BPEL workflow description is modified to solve a UNICORE 6/WS-BPEL workflow engine mapping problem. Details about this problem and the solution are described in Section 3.3.
4. The workflow is deployed to the WS-BPEL workflow engine. This is done by transferring the WS-BPEL workflow description and the corresponding deployment descriptor to the WS-BPEL engine by an appropriate adapter, in our case an ActiveBPEL adapter.
5. The corresponding Workflow Service (see Section 3.2) is created and registered to the UNICORE/X service container.

Subsequently, undeployment is executed as follows:

1. The factory service of the corresponding Workflow Service (see Section 3.2) is deregistered and removed from the UNICORE/X service container to prevent the creation of new Workflow Service instances.
2. The undeployment process waits for the termination of active workflow instances. The initiator of undeployment may decide whether these instances shall terminate normally (expiration date is infinite), instantly (expiration date is *0*), or at a specific date (expiration date is specified either explicitly or by a grace time). Except for normal termination, Workflow Service instance termination is enforced by the undeployment process at the given expiration date. By default, the normal termination strategy is used.
3. The actual Workflow Service (see Section 3.2) is deregistered and removed from the UNICORE/X service container.

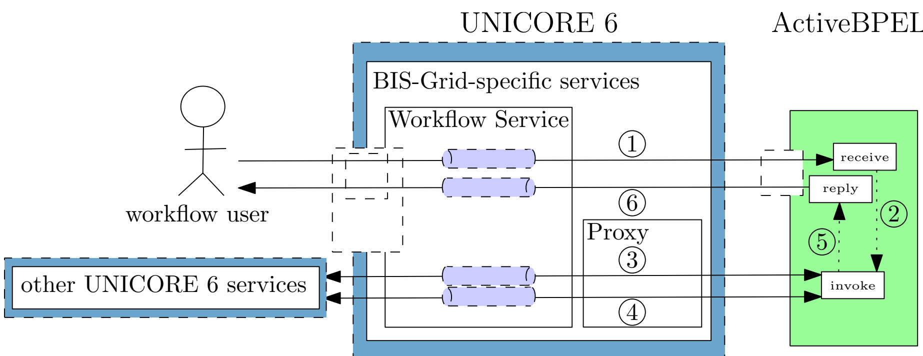


Fig. 3. Workflow Service Architecture and Example Usage

4. The WS-BPEL workflow is undeployed from the WS-BPEL workflow engine by using an appropriate adapter (ActiveBPEL). It has to be ensured that all data concerning the WS-BPEL workflow to be undeployed is removed.
5. The BIS-Grid Workflow Deployment Package and all related data are removed from the respective local file space.

Beside the functionalities described in this section, WS-BPEL engine management, and high-level monitoring and auditing of Workflow Service instances can also be seen as a part of workflow management. Appropriate services are already envisioned in our architecture but are not part of this work.

3.2 Workflow Service

A Workflow Service represents the execution of a specific deployed workflow. It is a WSRF service which instances are created through a corresponding factory service. Each instance is mapped directly to one workflow instance in the ActiveBPEL workflow engine. The Workflow Service must fulfil the use case shown in Figure 2(b): workflow instance creation, workflow execution, status information providing, and online configuration modification.

Figure 3 shows a more detailed view on a Workflow Service (note that the factory service is omitted). We assume a simple workflow that calls an external service and sends the response back to the user. Only relevant WS-BPEL activities are shown in the ActiveBPEL engine. Before the execution starts, the user has to create a new Workflow Service instance via the corresponding factory service.

The WSDL interface of this Workflow Service is a combination of two interfaces: The first one is the Workflow Service interface that offers BIS-Grid-specific operations, e.g. requesting information on workflow progress, changing/adding security tokens, or modifying the security policy. The second interface is the original workflow interface provided by the WS-BPEL engine integrated in the combined WSDL so that all service calls can be done on the Workflow Service.

Internally, the UNICORE/X service container contains an XFire SOAP engine[11] that processes incoming messages through an XFire Handler Pipeline. The last handler of the incoming pipeline is the so-called *Invoker* that manages the Java method invocation on the actual UNICORE/X service. The Invoker of a Workflow Service instance differentiates between BIS-Grid-specific service calls and service calls specific to workflow execution. Regarding the latter, the Invoker forwards the message to a general method. Grid-specific information such as security credentials or accounting and billing information is removed in a second configurable handler pipeline, thus converting the message to a standard message used in Web Service calls (see Figure 3 (1)). This will be realised using a separate handler pipeline. The BPEL workflow engine works up the message (2).

Vice versa, messages sent from the WS-BPEL workflow to invoke external services (3) are caught by a *proxy* also located in UNICORE/X. The proxy reads an identifier consisting of the name of the workflow and the id of the corresponding Workflow Service instance (cp. Section 3.3) into the message header, and forwards the message to the correct Workflow Service instance for further processing. By using another handler pipeline that depends on the current configuration of the Workflow Service instance, Grid-specific XML fragments - e.g. SAML assertions [9] - are added to the message and then forwarded through a certificate-secured SSL channel. In case of a synchronous service call the answer is subsequently processed in reverse, removing and processing Grid-specific XML fragments and forwarding it to the WS-BPEL engine through the connection held by the proxy (4). Regarding asynchronous service calls the connection will be closed after the message is sent.

The response message from the external service is also worked up by the BPEL engine (5) and the answer is send back to the user (6).

ActiveBPEL runs in a separate environment. It is not possible to access the engine without using the BIS-Grid-specific services of a UNICORE 6 installation. This is crucial for Grid workflow execution security since workflow access is controlled by UNICORE 6 security mechanisms using Grid credentials. All message transfers are performed using certificate-secured SSL channels.

3.3 UNICORE 6/WS-BPEL Engine Mapping

Using an arbitrary WS-BPEL engine, we have to deal with two different instances of the same workflow: a regular workflow instance in the actual WS-BPEL engine, and the WSRF Workflow Service instance in the UNICORE/X service container. It is the task of the UNICORE/X service extensions to check incoming and outgoing messages and to prepare them for Grid utilisation. To do so, it is necessary to map messages from the WS-BPEL engine to the correct WSRF instances and vice versa. In UNICORE 6, a Workflow Service instance is identified by a resource id which is also contained in its endpoint reference. Unfortunately, there is no identification information in the SOAP messages that are sent between UNICORE/X and the WS-BPEL engine which enables to

[11] http://xfire.codehaus.org/

conclude the original WS-BPEL instance. Such information is necessary for both synchronous and asynchronous external service invocations, e.g. since appropriate security credentials must be assigned to outgoing messages.

This problem is solved by modifying the WS-BPEL workflow description so that the resource identifier used by UNICORE/X is known to the WS-BPEL workflow and used in external message communication (cp. Section 3.1). Therefore, we stipulate that a WS-BPEL workflow expects the resource id of a corresponding Workflow Service instance to be attached to incoming messages that create workflow instances. Second, we stipulate an `assign` operation before each external service invocation within the WS-BPEL workflow description. This `assign` operation inserts the resource id into the message. All in all, the following requirements (RQ) must be met: For all *start*-messages[12] the corresponding XML schema must be extended by an extra variable for the resource id identifying the UNICORE/X Workflow Service instance (RQ1). There must be a dedicated process variable within the workflow to store the resource id during the whole process execution (RQ2). There must be an `assign` activity after each instance-creating activity which copies the resource id from the message into the process variable (RQ3). All outgoing message schemas, i.e. `reply` or `invoke`, must be extended by an extra variable to carry the resource id (RQ4). There must be an `assign` activity that copies the resource id from the dedicated process variable into the corresponding message variable before each activity that causes an outgoing message (RQ5). We developed WS-BPEL patterns that meet these requirements. They will be part of a WS-BPEL pattern catalogue that also contains patterns for Grid utilisation with standard WS-BPEL, for example.

3.4 Implementation Status

A first prototype of the BIS-Grid workflow engine is expected to be released in August 2008. The prototype and the WS-BPEL pattern catalogue mentioned in Section 3.3 will be made available on the BIS-Grid web site (www.bisgrid.de).

4 Conclusion and Future Work

We presented an overview on the architecture of the BIS-Grid workflow engine. It mainly consists of two Grid Services working together with the WS-BPEL engine ActiveBPEL. Instances of a Workflow Management Service hot-deploy Workflow Services that encapsulate workflows in the WS-BPEL engine, and propagate them as WSRF-compliant Grid Services. Within this architecture, neither the WS-BPEL engine nor the WS-BPEL 2.0 standard have to be adapted. The WS-BPEL engine is therefore exchangeable through any WS-BPEL-compliant engine. Also, the architecture offers further possibilities to easily adopt future features, e.g. by modifying handler pipelines.

[12] I.e. messages which are addressed to `receive` or `pick` activities with the `createInstance` attribute set to *yes*.

This paper presents a snapshot on our ongoing work. We recently started with the implementation, focussing on the basic functionalities, e.g. piping messages through the Workflow Service and on security issues. We will subsequently address human interaction in worflows, and consider an adequate workflow design tool in our future work.

References

1. Tim Banks. Web Services Resource Framework (WSRF) - Primer v1.2 (May 2006), http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf
2. Kuo-Ming, C., Younas, M., Griffiths, N., Awan, I., Anane, R., Tsai, C.-F.: Analysis of Grid Service Composition with BPEL4WS. In: Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA 2004), vol. 01, p. 284. IEEE Computer Society, Los Alamitos (2004)
3. Clementi, L., Cacciari, C., Melato, M., Menday, R., Hagemeier, B.: A Business-Oriented Grid Workflow Management System. In: Bougé, L., Forsell, M., Träff, J.L., Streit, A., Ziegler, W., Alexander, M., Childs, S. (eds.) Euro-Par Workshops 2007. LNCS, vol. 4854, pp. 131–140. Springer, Heidelberg (2008)
4. Dörnemann, T., Friese, T., Herdt, S., Juhnke, E., Freisleben, B.: Grid Workflow Modelling Using Grid-Specific BPEL Extensions (2007)
5. Emmerich, W., Butchard, B., Chen, L., Price, S.L., Wassermann, B.: Grid Service Orchestration Using the Business Process Execution Language (BPEL). Journal of Grid Computing, 283–304 (2006)
6. Gudenkauf, S., Hasselbring, W., Heine, F., Höing, A., Kao, O., Scherp, G.: A Software Architecture for Grid Utilisation in Business Workflows. In: MKWI. GITO-Verlag, Berlin (2008)
7. Gudenkauf, S., Hasselbring, W., Heine, F., Höing, A., Scherp, G., Kao, O.: Bis-Grid: Business Workflows for the Grid. In: CGW 2007 Proceedings, Krakow, Poland, pp. 86–94. ACC CYFRONET AGH (2008)
8. Leymann, F.: Choreography for the Grid: towards fitting BPEL to the resource framework: Research Articles. Concurr. Comput.: Pract. Exper. 18(10), 1201–1217 (2006)
9. Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., Scavo, T.: Security Assertion Markup Language (SAML) V2.0 Technical Overview (February 2007) (Working Draft), http://www.oasis-open.org/committees/download.php/22553/sstc-saml-tech-overview-2%200-draft-13.pdf
10. Slomiski, A.: On using BPEL extensibility to implement OGSI and WSRF Grid workflows: Research Articles. Concurr. Comput.: Pract. Exper. 18(10), 1229–1241 (2006)

Benchmarking of Integrated OGSA-BES with the Grid Middleware

Fredrik Hedman[1], Morris Riedel[2], Phillip Mucci[1], Gilbert Netzer[1], Ali Gholami[1], M. Shahbaz Memon[2], A. Shiraz Memon[2], and Zeeshan A. Shah[1]

[1] Center for Parallel Computers (PDC), Kungliga Tekniska Högskolan (KTH), SE-100 44 Stockholm, Sweden
[2] Jülich Supercomputing Centre, Forschungszentrum Jülich (FZJ), Leo-Brandt-Str. 1, Jülich, 52425, Germany

Abstract. This paper evaluates the performance of the emerging OGF standard OGSA - Basic Execution Service (BES) on three fundamentally different Grid middleware platforms: UNICORE 5/6, Globus Toolkit 4 and gLite. The particular focus within this paper is on the OGSA-BES implementation of UNICORE 6. A comparison is made with baseline measurements, for UNICORE 6 and Globus Toolkit 4, using the legacy job submission interfaces. Our results show that the BES components are comparable in performance to existing legacy interfaces. We also have a strong indication that other factors, attributable to the supporting infrastructure, have a bigger impact on performance than BES components.

Keywords: Performance analysis, Grid middleware, UNICORE, gLite, Globus Toolkit, OGSA-BES.

1 Introduction

Today's large-scale scientific research is supported by Grid and e-science infrastructures. Grids are composed of a set of heterogeneous resources that are managed by several interacting software components accessible as Grid services [1]. These infrastructures are increasingly often accessed via different flavors of Grid middleware technologies. Several different Grid infrastructures exist today. Some are primarily focused on maximized throughput, like EGEE, OSG and NGS; others are primarily driven by high-performance computing (HPC) needs, like DEISA and TeraGrid. With emerging Grid infrastructure interoperability, it is now becoming possible and realistic to combine these different types of resources for major scientific research areas that demand both high throughput and HPC to make progress [2]. In these scenarios, Grid middleware performance becomes important to measure and understand.

Recently, many Grid middleware technologies have been augmented with implementations of proposed recommendations from the Open Grid Forum (OGF). An example is the OGSA Basic Execution Service (BES) [3] for job management and submission. Adding BES to a middleware is motivated by the gain in interoperability between different Grids [4], and by the increase in performance and handling of jobs between these middleware technologies.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 113–122, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Performance analysis and benchmarking of Grid technologies in general, and Grid middleware in particular, is still an emerging area. Some results covering different approaches and tools can be found in [5,6,7].

In contrast to these approaches, we presented in [8] a "black-box" approach for analyzing Grid middleware, using a straightforward and non-invasive platform independent method. In this paper, we build on our previous work to assess the performance of the recently finalized BES implementations for three[1] different Grid middleware stacks: gLite, Globus Toolkit 4 and UNICORE 6. Performance analysis of the UNICORE implementation [9] becomes specifically important since it has been deployed on several DEISA sites for evaluations.

This paper is structured as follows. Section 2 gives a background and illustrates the design of our approach for BES. Section 3 provides implementation details of the benchmarked BES implementations. We present results from our performance measurements and provide insights by evaluating them with respect to the core technologies and job management handling within the different Grid middlewares in Section 4. Finally a conclusion is presented in Section 5.

2 Black-Box Benchmarks of Grid Middleware

From a user perspective Grid middleware adds a certain overhead compared to local execution. Independent of the size of the submitted job, fixed costs contribute to the overhead of the Grid system. It is important to be able to diagnose and address this overhead early in the design cycle of a new component. Grid developers can use this information for evaluating design and implementation trade-offs while Grid users can try to amortize the overhead by submitting fewer longer running jobs.

To analyze the performance of Grid middleware rather than just measuring the time an application spends on a resource, we have demonstrated a method to measure the time spent on the "grid work" per job in [8]. In this paper we build on our previous work and investigate the performance of recently developed alternative job submission components in different Grid middlewares. We assume that a component can be treated as a black box into which jobs are submitted and from which results are returned. The results obtained can be compared with our baseline measurements for consistency and rough estimates of the overhead introduced by the new components.

Features and capabilities provided by the investigated job submission interfaces vary. Thus we implemented three different variants of our basic benchmark. For the WS-GRAM baseline measurements we used the `globusrun-ws` command line client. It can submit and monitor one job at a time. Our implementation uses 10 parallel threads that first submit the required number of jobs and poll for their completion in a round-robin fashion. CondorG and CLIQ offer bulk submission capability: a single job submission followed by local polling is used. Finally

[1] Our work was done in the context of the OMII-Europe project which focused on these three Grid middleware platforms.

for the new BES and UNICORE 6 mechanisms we use a serial implementation, submitting and monitoring one job at a time.

To provide a basis for reproducible and repeatable results, it is necessary to create identical conditions when performing comparisons of Grid middleware. Our experience is that this can be very timeconsuming and in some cases not feasible. We claim a best effort approach can still give some comparable performance measurements across different Grid middleware.

3 OGSA-BES Implementations

BES provides easy, intuitive and standardized access to computational resources. The OMII-Europe project provided BES implementations for the three middleware stacks which we benchmarked.

The BES specification [3] defines two mandatory (`BESFactory` and `BESManagement`) and one optional (`BESActivity`) interface. The management interface allows to control the service itself. The factory interface provides job submission and bulk monitoring capabilities while the activity interface allows monitoring of a single job. Jobs submitted to the BES interface have to be described in the Job Submission and Description Language (JSDL) [10].

3.1 OGSA-BES Implementation of UNICORE 6

At the time of writing, UNICORE 6 offers two WS-based interfaces for job submission and management: BES and UNICORE Atomic Service (UAS) [9]. Both interfaces allow for job management and control functions using WS messages. Both interfaces are adopted within the middleware and deployed on top of resource management systems (RMS) (e.g. Torque, LoadLeveler) that in turn deal with job scheduling on computational resources.

Since BES allows for a flexible implementation strategy, the UNICORE developers have been able to directly use the interfaces of the execution back-end of UNICORE 6, which is the enhanced Network Job Supervisor (XNJS) [11]. The implementation consists of the mandatory `BESFactory` and `BESManagement` interfaces as well as the optional `BESActivity`. In comparison to BES, the UAS is a suite that additionally provides storage management and file transfer mechanisms that have been leveraged by the BES implementation. UAS also accepts jobs described in JSDL.

3.2 OGSA-BES Implementation of gLite

The integration of the BES interface into the gLite middleware was accomplished by extending the CREAM computing element which provides both the back-end core and the legacy interface. The new interface is implemented as a separate plugin on top of CREAM core. Both interfaces can share the same core so that jobs can be submitted to the same resource via both interfaces.

3.3 OGSA-BES Implementation of Globus

The BES service for the Globus toolkit is implemented as a thin wrapper layer in front of the WS-GRAM job submission service. The legacy WS-GRAM interface is also available to support the existing Globus infrastructure.

The BES service itself is stateless with the exception of a single flag that allows to stop job submission via the management interface. Each incoming request is translated into corresponding WS-GRAM requests, which involves a translation of the jobs description from JSDL to RSL needed by the WS-GRAM. This adds some overhead to each BES call but allows for a transparent deployment of BES interfaces into existing Globus infrastructures, since the BES service simply behaves like a WS-GRAM client and can even be in a completely different organizational domain.

4 Benchmark Results

Using the “black-box” approach described in [8] we have collected baseline performance data for a number of Grid middlewares. We use the baseline data to assess the performance of the new BES components.

4.1 Baseline Results

The computer platform used consisted of nodes with a 2.8 GHz Intel Pentium D CPU and 2 GB RAM connected through gigabit Ethernet and running Scientific GNU/Linux version 4 operating system. Even though the test-bed was a controlled environment with only local network traffic and consistent operating system and middleware installations, big variations between different runs were detected. This underlines the importance of a carefully controlled environment to arrive at repeatable and comparable experiments for quantitative analysis.

Figures 1 and 2 show a selection of our results from a number of benchmark runs for Globus Toolkit 4 and UNICORE 5/6 in our test-bed. Keeping the large variation in mind a number of qualitative conclusions can be drawn. For Globus Toolkit middleware we conclude:

- Submission via the CondorG interface yields best results for the workload of the benchmark. This is also the mechanism that is recommended by the Globus Alliance for the case of submitting a large number of jobs.
- Data staging has by far the biggest influence on job submission performance. The fastest runs occur for all tested submission mechanisms when staging of input and output is not used.
- The results for the WS-GRAM submission tests show that considerable stress is put on both the middleware and the submission script. For instance, the two runs with the longest run time in Fig. 1 show that about half of the jobs finished almost simultaneously which is most likely an artifact of the polling method used by the benchmark driver to detect job completion. The polling attempts load both the middleware and the benchmark driver and this load is highest in the beginning of the run when there are many unfinished jobs that need to be checked.

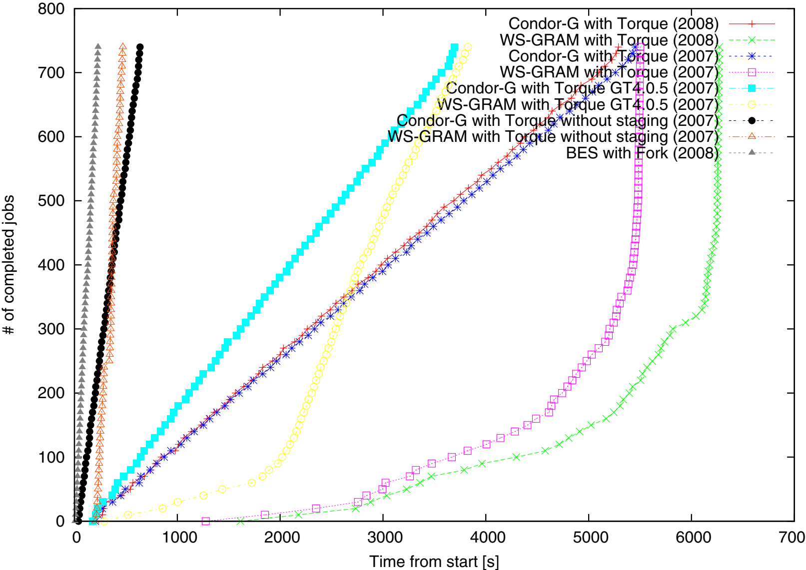


Fig. 1. Selected system level performance runs for the Globus job submission components. The data sets show that different job submission mechanisms (WS-GRAM, CondorG, BES) show different characteristics, but also that staging effort dominates over all other issues. BES does not do any staging and is shown for reference.

Fig. 2 shows the results from the evaluation of the system level performance for the UNICORE 6 middleware stack, including comparisons with UNICORE 5:

- The UNICORE 5 results show consistent behavior for both Torque and Fork local resource management. In this cases we used the batch submission capability provided by the CLIQ client.
- The results show also that the scheduling policy of the local resource manager (batch system) can have an impact on the total performance. This is exemplified by the results obtained when using the MAUI scheduler for the Torque batch system.
- The UNICORE 6 system level performance shows considerable spread in contrast to the UNICORE 5 results. Part of this could be attributed to the fact that different versions of the middleware were used to make the measurements. However the large spread between the Fork and Torque performance points more to the sensitivity of the UNICORE 6 benchmarks towards small changes in the benchmarking environment. This is probably caused by the fact that the current benchmark tools are restricted to serial submission when targeting UNICORE 6 which puts more emphasis on the latency of a single submission.
- In contrast to the Globus Toolkit, no surge of jobs completing can be seen. This leads to the conclusion that the selected benchmark method is well

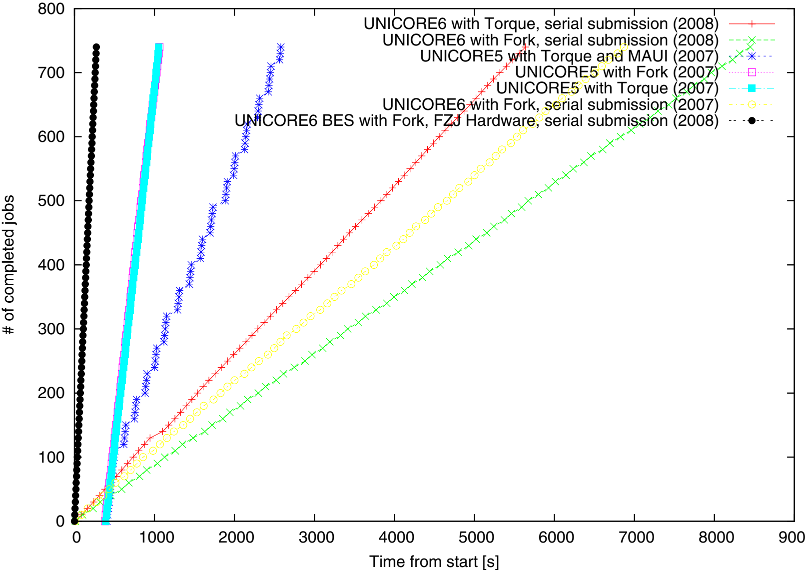


Fig. 2. System level performances for the UNICORE middleware stack. The data-sets shown are both for version 5 and 6 of UNICORE. As a reference the results for the BES submission are also given.

adopted towards the UNICORE middleware stack thus not exhibiting the polling problems of the Globus tools.

4.2 BES Components Performance

With the availability of the BES enhanced job submission components from the OMII-Europe [12] project, we have performed a first evaluation of the performance of these components. Since these components are handling job submission we compare them against the system level performance evaluations. It is however essential to bear in mind that no firm comparisons can be made because of differences in the platforms and setups that were used. A number of adoptions had to be made in order to conduct the performance evaluation at this early stage of deployment of the BES components. In particular the following differences to the system level evaluation are present:

– Because of the limited time we used the test services provided by developers of the respective BES components. This means that the hardware and software setup differs significantly. This is most apparent in the case of the gLite test service which uses a complete batch system and computing element (the CREAM-CE) for its backend compared to the simple process forking backends used by the Globus Toolkit and UNICORE services. The specification of hardware and software for BES endpoints used are:

CREAM/BES. The `omiivm03.cnaf.infn.it` host is deployed as a virtual machine on the omii005 host. This machine has 2 Intel(R) Xeon(R) 2.00GHz CPUs and 4 GB RAM. The machine hosts four virtual machines each with an equal share of the available resources of which `omiivm03` is one.

GT/BES. The GT/BES server is hosted on `romana.pdc.kth.se` and has a Intel Dual Core 2.13 GHz CPU with 1 GB memory running Gentoo GNU/Linux 2.6.

UNICORE/BES. The BES server hosted on `zam461.zam.kfa-juelich.de` has a Intel Dual Xeon 3 GHz CPU with 2 GB memory running SuSE GNU/Linux 9.3.

Benchmark client. An Intel Dual Core CPU 2.13 GHz running Gentoo 2.6 machine targeting different endpoint. The cost of a ping to the endpoints were approximately 30 ms.

- No attempts at handling input or output data (data staging) for the job were made.
- The current BES benchmark does not submit jobs in parallel but processes them in a serial fashion.

In total, the BES benchmark is considerably simpler in nature than the earlier system level benchmarks. Despite this fact, we found it helpful in uncovering a few interesting facts about the behavior of the OMII-Europe BES implementations:

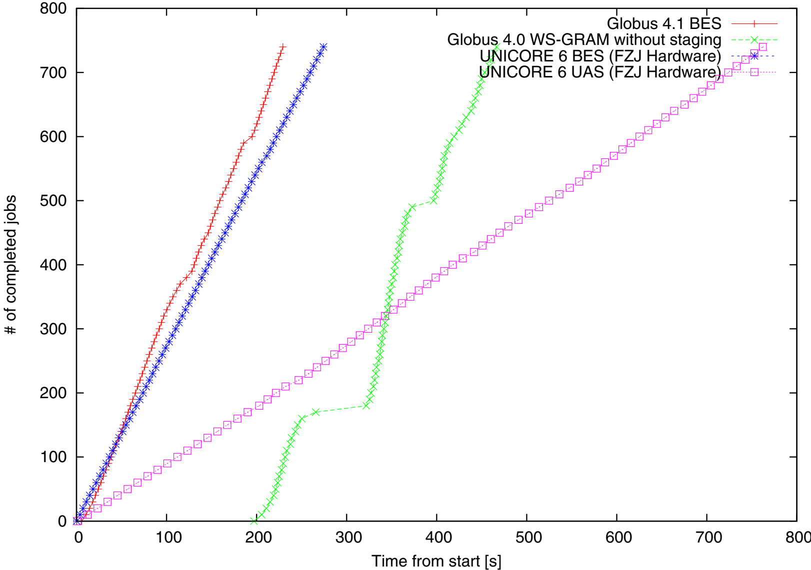


Fig. 3. Performance of Globus and UNICORE 6 BES implementations compared to the UNICORE 6 component performance for the legacy interface and the system level performance of Globus legacy WS-GRAM without any staging. The figure shows the number of jobs that have been completed versus the time from the start of the experiment.

– We were able to make a close comparison between the UNICORE 6 BES adoption and the legacy interface (i.e. UAS) using the same method. The legacy system seemed to exhibit 3 times higher latencies than the BES system (Fig. 3). The benchmark that we used however emphasizes the latency component because of the serial nature of the submission. Preliminary investigations show that this was caused by a polling rate limitation in the UAS client.
– Despite the differences in hardware and web services implementation, the performance of the UNICORE and Globus Toolkit implementations is about the same and comparable to the performance of the legacy Globus WS-GRAM when no data staging is done. This can be seen in Fig 3. The increased delay until completion of the first job in the system level script is caused by the concurrent submission of all 750 jobs in the beginning of the benchmark run.
– The impact of staging and logging is far bigger than any overhead introduced by the BES services. This becomes apparent in Fig 4 where the gLite BES implementation is compared to the legacy Globus WS-GRAM performance with and without staging. The inclusion of input/output staging into the WS-GRAM submission increased total execution times by a factor of roughly 6. Also the job completion rate in the early stages of the WS-GRAM experiment match the rate from the gLite BES. This is a strong indication that the performance of the gLite service is comparable to the other BES implementations with a more realistic backend and job setup where input and output

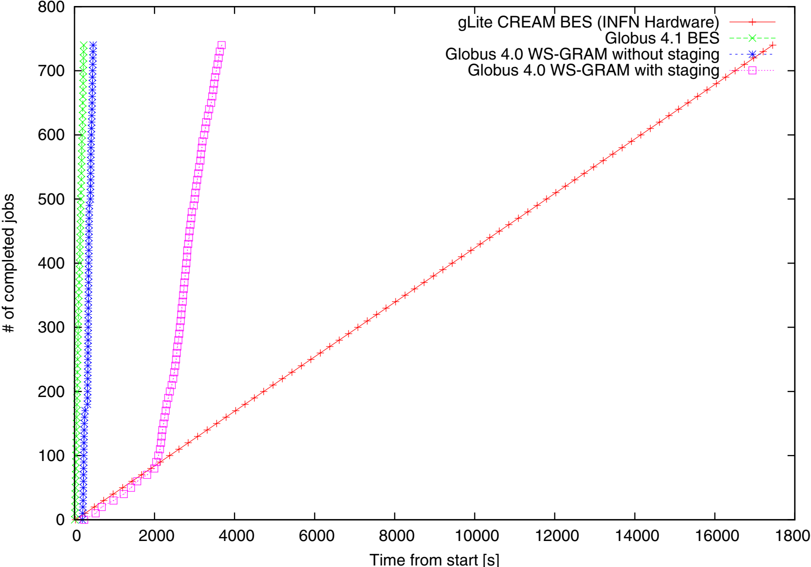


Fig. 4. Performance of the gLite BES implementation compared to the Globus BES implementation and Globus legacy WS-GRAM system level performance with and without staging. It is important to notice that the gLite CREAM BES uses a full batch system backend.

would be transferred to the user and/or permanent storage. Also, the lack of concurrency may slow down the gLite BES since logging phases cannot overlap with execution of the next job.

To sum up, the simple evaluation of the performance of the BES services does show that the performance of the BES services should be acceptable compared to the performance of the legacy job submission mechanisms. However, further targeted investigation in carefully controlled environments would be necessary to conclusively asses the performance of these new mechanisms.

5 Conclusion

A necessary prerequisite to benchmark a piece of software is the ability to execute this software under controlled and well known conditions. In fact, the BES components are dependent on local resource management systems to perform the actual job execution. As the presented "black-box" results suggest, configuration and services, for example handling of input and output data or logging, carried out by these "back-end" infrastructure have a large impact on the measured performance of the component. We tried to resolve this issue by estimating the effect of the back-end system onto the component performance by using alternate legacy components that utilized the same back-end infrastructure and use them as a baseline for relative comparison. We used this approach for the BES job submission benchmark provided in this paper.

In summary, our results show that the performance of the BES components that were evaluated are comparable to existing legacy solutions. Different security setups of the components may also lead to different performance, but in this paper we clearly consider them out-of-scope. The "black-box" experiments strongly indicate that other factors attributable to the supporting infrastructure have a bigger impact on performance than the use of BES components.

Acknowledgment

This work is supported by the European Commission through the OMII-Europe project-INFSO-RI-031844. Further information see [12].

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Elsevier, Amsterdam (2004)
2. Riedel, M., et al.: Improving e-Science with Interoperability of the e-Infrastructure EGEE and DEISA. In: 31st International Convention MIPRO, Conference on Grid and Visualization Systems (GVS), Opatija, Croatia (May 2008) (accepted)
3. Foster, I., Grimshaw, A., Lane, P., Lee, W., Morgan, M., Newhouse, S., Pickles, S., Pulsipher, D., Smith, C., Theimer, M.: OGSA Basic Execution Service Version 1.0. Technical report, Open Grid Forum (2007), `http://www.ogf.org/documents/GFD.108.pdf`

4. Marzolla, M., Andreetto, P., Venturi, V., Ferraro, A., Memon, A., Memon, M., Twedell, B., Riedel, M., Mallmann, D., Streit, A., van de Berghe, S., Li, V., Snelling, D., Stamou, K., Shah, Z., Hedman, F.: Open Standards-based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE. In: Proceedings of International Interoperability and Interoperation Workshop (IGIIW) 2007 at 3rd IEEE International Conference on e-Science and Grid Computing, Bangalore, India, pp. 592–599. IEEE Computer Society, Los Alamitos (2007)
5. Dikaiakos, M.: Grid benchmarking: Vision, challenges, and current status. Concurrency and Computation: Practice and Experience 19(1), 89–105 (2007)
6. Nemeth, Z., Gombas, G., Balaton, Z.: Performance evaluation on grids: Directions, issues and open problems. In: 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2004), p. 290 (2004)
7. Snavely, A., Chun, G., Casanova, H., der Wijngaart, R.F.V., Frumkin, M.A.: Benchmarks for grid computing: a review of ongoing efforts and future directions. SIGMETRICS Perform. Eval. Rev. 30(4), 27–32 (2003)
8. Alexius, P., Elahi, B.M., Hedman, F., Mucci, P., Netzer, G., Shah, Z.A.: A Black-Box Approach to Performance Analysis of Grid Middleware. In: Bougé, L., Forsell, M., Träff, J.L., Streit, A., Ziegler, W., Alexander, M., Childs, S. (eds.) Euro-Par Workshops 2007. LNCS, vol. 4854, pp. 62–71. Springer, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-78474-6_10
9. Riedel, M., Schuller, B., Mallmann, D., Menday, R., Streit, A., Tweddell, B., Memon, M., Memon, A., Demuth, B., Lippert, T., Snelling, D., van den Berghe, S., Li, V., Drescher, M., Geiger, A., Ohme, G., Benedyczak, K., Bala, P., Ratering, R., Lukichev, A.: Web Services Interfaces and Open Standards Integration into the European UNICORE 6 Grid Middleware. In: Proceedings of 2007 Middleware for Web Services (MWS 2007) Workshop at 11th International IEEE EDOC Conference The Enterprise Computing Conference. IEEE Computer Society, Los Alamitos (2007)
10. Anjomshoaa, A., et al.: Job Submission Description Language (JSDL) - Specification Version 1.0. Technical report, Open Grid Forum (2005), http://www.ogf.org/documents/GFD.56.pdf
11. Schuller, B., Menday, R., Streit, A.: A versatile execution management system for next-generation UNICORE grids. In: Lehner, W., Meyer, N., Streit, A., Stewart, C. (eds.) Euro-Par Workshops 2006. LNCS, vol. 4375, pp. 195–204. Springer, Heidelberg (2007)
12. Open Middleware Infrastructure Institute for Europe. Project no: RI031844–OMII-Europe, http://omii-europe.org

Second Workshop on Highly Parallel Processing on a Chip (HPPC 2008)

It is now noncontroversial to assume that explicit, general-purpose on-chip parallelism will play a much more prominent role in future computing systems, ranging from embedded and portable systems, through servers, to compute farms and high-performance systems. What is, on the other hand, currently completely open is how such systems will be organized from the architectural point of view, how parallelism shall be exposed to the programmer (and user), as well as the nature of the parallelism (general or special purpose) that can or will be made available.

The Workshop on Highly Parallel Processing on a Chip (HPPC) takes as premise that the drive for increased application performance, technological feasibility and technological and environmental constraints (for instance on power consumption) within the next 5 to 10 years will lead to chips with a significant number (read: more than four or eight) of general-purpose cores, and/or possibly an order of magnitude more special-purpose cores, quite likely complemented by new and/or different paradigms for on-chip communication and memory organization than the processors that are now on the market. These developments pose major challenges to architecture, language and compiler design, algorithms, and application developments, in order to fully (or acceptably) exploit the raw, provided compute power. The HPPC workshop aims to be a forum for discussion of such fundamental issues. It is open to all aspects of existing and emerging, envisaged and imagined multi-core (by which is meant: many-core) chips with a *significant amount of parallelism*, especially to considerations on novel paradigms and models and the related architectural and linguistic support. To be able to relate to the parallel processing community at large, which we consider essential, the workshop is organized in conjunction with Euro-Par, the main European (but international) conference on all aspects of parallel processing.

For HPPC 2008, the second installment of the workshop, 6 papers were selected for presentation and subsequent publication out of the 20 submissions received in response to the call for papers that was launched early in 2008. The submitted papers were all relevant to the workshop themes, some more than others, and due to the limited time for the workshop (an extended half-day event), only about 30% of the submissions could be accepted. The workshop organizers thank all contributing authors, and hope that they will also find it worthwhile to submit contributions next year. All contributions received *four* reviews by members of the Program Committee, who are likewise all thanked for the time and expertise they put into the reviewing work, and for getting it done within the rather strict time limit. The final decision on acceptance was made by the Program Chairs based on the recommendations from the Program Committee.

The Euro-Par 2008 workshop day was lively and well-organized, and the HPPC workshop had a cumulative attendance of more than 40. In addition to the six contributed talks, the workshop had two longer, invited talks by Gianfranco

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 123–125, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Bilardi (on "Models for Parallel and Hierarchical On-Chip Computation") and Chris Jesshope (on "Building a Concurrency and Resource Allocation Model into a Processor's ISA"). The HPPC 2008 workshop organizers thank all attendees, who contributed much to the workshop with questions, comments and discussion, and hope they found something of interest in the workshop, too. We also thank the Euro-Par organization for creating the opportunity to arrange the HPPC workshop in conjunction with the Euro-Par conference, and of course all Euro-Par 2008 organizers for their help and (excellent) support both before and during the workshop. Our sponsors VTT, NEC Laboratories Europe and Euro-Par 2008 are warmly thanked for the financial support that made it possible to invite Gianfranco Bilardi and Chris Jesshope, both of whom we sincerely thank for accepting our invitation to speak and for their excellent talks.

These proceedings include the final versions of the presented HPPC papers (as a matter of principle, accepted papers not presented at the workshop are not included in the proceedings), taking the feedback from reviewers and workshop audience into account. In addition to the reviews by the Program Committee prior to selection, an extra, post-workshop (blind) "*reading*" of each presented paper by one of the other presenters was introduced with the aim of getting fresh, uninhibited high-level feedback for the authors to use at their discretion in preparing their final version (no papers would have been rejected at this stage – bar major flaws). This idea will be continued for HPPC 2009.

The contributed papers are printed in the order they were presented at the workshop. The abstracts of the two invited talks by Gianfranco Bilardi and Chris Jesshope have also been included in the proceedings. Thematically the contributed papers cover aspects of language and algorithms support for an actual, well-known, heterogeneous multi-core processor ("Optimized Pipelined Parallel Merge Sort on the the Cell BE" by Keller and Kessler, and "Compile-Time and Run-Time Issues in an Auto-parallelization System for the Cell BE Processor" by Donaldson, Keir and Lokhmotov), general, run-time support for heterogeneous multi-cores ("A Unified Runtime System for Heterogeneous Multi-core Architectures" by Augonnet and Namyst), improvements of software implementations of the transactional memory programming model ("Adaptive Read Validation in Time-Based Software Transactional Memory" by Atoofian, Baniasadi and Coady), high-level, general-purpose, programming support tools ("Towards an Intelligent Environment for Programming Multi-core Computing Systems" by Pllana et al.), as well as considerations on power-constrained limits to scaling of multi-cores ("(When) Will CMPs Hit the Power Wall?" by Meenderinck and Juurlink). The last mentioned paper, very much aligned with the premise of the HPPC workshop, conjectures chip-multiprocessors with 999 cores by 2022.

The HPPC workshop is planned to be organized again in conjunction with Euro-Par 2009.

November 2008

Martti Forsell
Jesper Larsson Träff

Sponsors

VTT, Finland	http://www.vtt.fi
NEC Laboratories Europe, Germany	http://www.it.neclab.eu
Euro-Par	http://www.euro-par.org

Models for Parallel and Hierarchical On-Chip Computation

Gianfranco Bilardi

Department of Information Engineering
University of Padova
Italy
bilardi@dei.unipd.it

Abstract. Chip Multiprocessors have the potential to deliver significant performance, easily in the Teraflop/s range within a few years. To achieve the full exploitation of this potential, it is crucial to develop adequate models of computation that can guide the optimization of algorithms and of architectures. This talk will present results and open issues along three directions:

1. The pipeline of accesses in the memory hierarchy to increase memory bandwidth utilization.
2. The network-oblivious approach as a step toward efficient algorithmic portability across chip multiprocessors with different organizations.
3. The information-exchange methodology to identify the best partition of chip area between functional units and storage elements, under chip I/O bandwidth constraints.

Short Biography

Gianfranco Bilardi is Professor of Computer Science at the University of Padova, Italy. He also holds an Academic Visitor position with IBM Research, at the T. J. Watson Laboratory. Previously, he has been an Assistant Professor at Cornell University, New York, USA. He has visited several other institutions, including the University of California at Berkeley and at Irvine, Brown University, the University of Illinois at Chicago, and the Max Planck Institut für Informatik in Saarbrücken, Germany.

Bilardi's research is mostly in the areas of parallel algorithms and architectures, high performance computing, VLSI, and signal processing. This research has been sponsored by various agencies and companies, both in Europe and in the USA. He has coordinated a number of research projects, at the national and at the European level. Currently, he is the Coordinator of the Center of Excellence MIUR "Science and Applications of Advanced Computing Paradigms," established in 2001, and involving researchers in different areas of informatics and computational sciences and engineering. He has (co)authored over 75 international publications.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 127–128, 2009.
© Springer-Verlag Berlin Heidelberg 2009

He is part of the executive committee of the Bertinoro International Center for Informatics (BiCi), which hosts and partially sponsors around twenty events per year, between workshops, conferences, and schools. He has served on many program committees of international conferences, editorial boards, and research evaluation committees. With P. Pattnaik, he has established and coordinated the workshop series "ScalPerf: Scalable Approaches to High Performance and High Productivity Computing", whose 6th edition is planned for September 2008.

Building a Concurrency and Resource Allocation Model into a Processor's ISA

Chris Jesshope

Computer Systems Architecture Group, Informatics Institute
University of Amsterdam
The Netherlands
jesshope@science.uva.nl

Abstract. We are now facing the prospect of no increases in computer system's performance unless we harness and efficiently exploit the concurrency that comes from multiple cores on a chip. It should be emphasised that the issues in exploiting concurrency are scale invariant and relate to a few simple parameters and issues. These are: the ratio of the throughput of computation and communication (both local and global), which determines how computation can be distributed and the cost of concurrency creation compared to computation. The latter determines the grain size of the computation. Finally we need virtual concurrency or parallel slack and an efficient data-driven scheduling mechanism, in order to tolerate the latency in any asynchronous activity in the computation, such as access to remote data and resource sharing. The concurrency model used must also be well behaved, i.e. provide determinism of the values computed (although not necessarily of the time required to compute them) and have safe composition. Although these concurrency issues are scale invariant it makes sense to implement them at the lowest scale possible, i.e. at the level of machine instructions, which have overheads measured in single cycles. In this way, all levels of concurrency may be exploited, which is important when dealing with legacy or constrained code. This presentation will explore work undertaken at the University of Amsterdam in designing and evaluating micro-grids of micro-threaded processors that meet these requirements. Moreover the concurrency model developed in this work, SVP, is free of deadlock under composition and has built into its implementations issues which are considered to be operating system ones. Namely it builds in the abstraction of a place, which capture resources and security both in using places and in controlling the execution of concurrency at a place. As the implementation of the concurrency model also manages mapping and scheduling of concurrency, it can truly be said that SVP is an operating system kernel built into the ISA of the processor.

Short Biography

Chris Jesshope is Professor of Computer Systems Engineering at the University of Amsterdam and has held this post since 2004. Prior to this, he has held

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 129–130, 2009.
© Springer-Verlag Berlin Heidelberg 2009

posts in a number of universities including a Readership at Southampton University and a Chair at Surrey University, two of the top Electronic Engineering schools in the UK. Professor Jesshope is a Chartered Engineer, a Fellow of the BCS, a Member of the IEEE and a Member of the IEEE Computer Society. His professional activities have included membership on various funding agency committees in both the UK and the Netherlands and the prestigious post of Editor of the IEE Proceedings part E (Computers and Digital techniques) over a 10 year period. He has been involved in numerous program committees, is the Founding Chair of the steering committee of the Microgrid International Workshop on on-chip concurrency and was the founding chair of the steering committee for the Euro-Par International Conference. He has also been general chair for nine international conferences and workshops. Professor Jesshope has given in excess of 50 invited papers or keynote presentations in his career, has written or edited 17 major works, including the very successful book Parallel Computers and published in excess of 160 refereed papers. Most of this work has been in the field of parallel computer architectures and concurrent programming.

Optimized Pipelined Parallel Merge Sort on the Cell BE

Jörg Keller[1] and Christoph W. Kessler[2]

[1] FernUniversität in Hagen, Dept. of Math. and Computer Science, 58084 Hagen, Germany
[2] Linköpings Universitet, Dept. of Computer and Inf. Science, 58183 Linköping, Sweden

Abstract. Chip multiprocessors designed for streaming applications such as Cell BE offer impressive peak performance but suffer from limited bandwidth to off-chip main memory. As the number of cores is expected to rise further, this bottleneck will become more critical in the coming years. Hence, memory-efficient algorithms are required. As a case study, we investigate parallel sorting on Cell BE as a problem of great importance and as a challenge where the ratio between computation and memory transfer is very low. Our previous work led to a parallel mergesort that reduces memory bandwidth requirements by pipelining between SPEs, but the allocation of SPEs was rather ad-hoc. In our present work, we investigate mappings of merger nodes to SPEs. The mappings are designed to provide optimal trade-offs between load balancing, buffer memory consumption, and communication load on the on-chip bus. We solve this multi-objective optimization problem by deriving an integer linear programming formulation and compute Pareto-optimal solutions for the mapping of merge trees with up to 127 merger nodes. For mapping larger trees, we give a fast divide-and-conquer based approximation algorithm. We evaluate the sorting algorithm resulting from our mappings by a discrete event simulation.

1 Introduction

Multiprocessors-on-chip are about to become the typical processors to be found in desktops, notebooks and clusters. Besides multicores based on x86 architectures, we also find new designs such as the Cell Broadband Engine processor with 8 parallel processors called SPEs and a Power core (see e.g. [1] and the references therein). Currently, explicit parallel programming is necessary to exploit the raw power of these processors. Many applications use the Cell BE like a dancehall architecture, i.e. all SPEs load data from the external memory, and use their small local memories (256 KB for code and data) as explicitly-managed caches. Yet, as the bandwidth to the external memory is the same as each SPE's bandwidth to the element interconnect bus (EIB) [1], the external memory limits performance and prevents scalability. Bandwidth to external memory is a common bottleneck in multiprocessors-on-chip, and the increasing number of cores will intensify the problem [2]. A scalable approach to parallelism on such architectures therefore must use communication between the SPEs to reduce communication with external memory.

Sorting is an important subroutine in applications ranging from computational geometry to bio informatics and data bases. Parallel sorting algorithms on a wealth of architectures have therefore attracted considerable interest continuously for the last decades,

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 131–140, 2009.
© Springer-Verlag Berlin Heidelberg 2009

see e.g. [3,4]. As the computation to memory-transfer ratio is quite low in sorting, it presents an interesting case study to develop bandwidth efficient algorithms.

Sorting on the Cell BE presents several challenges. First, the SPEs' local memories are so small that most parallel sorting algorithms must mainly use the external memory, and thus will not be memory-efficient. Algorithms which do not suffer from this problem must also have very simple, data-independent control structures that are able to efficiently use the SPEs' SIMD structure and minimize branching. Sorting algorithms implemented for the Cell BE [5,6] use bitonic sort or merge sort and work in two phases to sort a data set of size n with local memories of size n'. In the first phase, blocks of data of size $8n'$ that fit into the combined local memories of the 8 SPEs are sorted. In the second phase, those sorted blocks of data are combined to a fully sorted data set. We concentrate on the second phase as the majority of memory accesses occurs there. In [5], this phase is realized by a bitonic sort because this avoids data dependent control flow and thus fully exploits the SIMD architecture of the SPEs. Yet, $O(n \log^2 n)$ memory accesses are needed, and the reported speedups are small. In [6], mergesort with 4-to-1-mergers is used in the second phase, where the mergers use bitonic merge locally. The data flow graph of the merge procedures thus forms a fully balanced quad-tree. As each SPE reads from main memory and writes to main memory, all n words are transferred from and to main memory in each round, resulting in $n \log_4(n/(8n')) = O(n \log_4 n)$ data being read from and written to main memory. While this improves the situation, speedup still is limited.

In order to overcome this bottleneck, we propose to run merger nodes belonging to consecutive layers of the merge tree concurrently, so that output from one merger is not written to main memory but sent to the SPE running the follow-up merger node, i.e. we use a form of pipelining. If we can embed k-level b-ary merge trees in this way, we are able to realize parallelized b^k-to-1 merge routines and thus increase the ratio of computation to memory transfer by a factor of $k \cdot \log_4 b$. Yet, this must be done such that all SPEs are kept busy. As in [6], a merger node does not process complete blocks of data before forwarding its result block, but uses fixed sized chunks of the blocks, i.e. a merger node is able to start work as soon as it has one chunk of each of its input blocks, and as soon as it has produced one chunk of the output block, it forwards it to the follow-up node. This form of streaming allows the use of fixed size buffers, holding one chunk each. To overlap data transfer and computation, the merger nodes should use double buffering at least for their inputs, and the buffers should have a reasonable minimum size to allow for efficent data transfer between SPEs.

Both [6] and our approach may benefit from a sample sort [7] preprocessing to reduce the problem to p sorts of cn/p data each, where $c \leq 3$ with high probability, which avoids $\log_4 p$ and $\log_{b^k} p$ rounds, respectively.

Ensuring that our pipeline runs close to the maximum possible speed requires load balancing. If a merger node u must provide an output rate of τ words per time unit, then the mergers u_i, where $1 \leq i \leq b$, feeding its inputs must provide a rate of τ/b words per time unit on average. However, if the values in the output chunk produced by u_i are much larger than those in u_j (see Fig. 1), u will only process values from u_j for some time, so that u_j must produce at a double rate for some time, while u_i will be stalled because of finite buffering between u_i and u. Otherwise the rate of u will reduce.

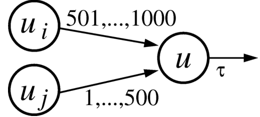


Fig. 1. Load balancing between merger nodes

Finally, the merger nodes should be distributed over the SPEs such that not all communication between merger nodes leads to communication between SPEs, in order not to overload the EIB.

The remainder of this article is organized as follows. In Section 2, we present the mapping problem sketched here in a formal way, give an integer linear programming solution to compute an optimal mapping of a b-ary merge tree onto the SPEs for small and medium-sized merge trees, and present an approximation algorithm based on divide-and-conquer. In Section 3, we discuss how our mapping turns into an efficient sorting algorithm, and we present simulation results. Section 4 concludes.

2 Mapping Trees onto Processors

2.1 Definitions

Given is a set $P = \{P_1, \ldots, P_p\}$ of p processors interconnected by a ring, and a k-level balanced b-ary tree $T = (V, E)$ directed towards its root, to be mapped onto the processors. Information in the tree flows from the leaves towards the root, input being fed in at the leaves and output leaving the tree root. Each node v in the tree processes b designated incoming data streams and combines them into one outgoing data stream of rate $0 < \tau(v) \leq 1$. Hence, the incoming data streams on average will have rate $\tau(v)/b$, if we assume finite buffering within nodes.

The *computational load* $\gamma(v)$ that a node v places on a processor that it is mapped to is proportional to its output rate $\tau(v)$, hence $\gamma(v) = \tau(v)$. The tree root r has a normalized output rate of $\tau(r) = 1$. Thus, each node v on level i of the tree, where $0 \leq i \leq k-1$, has $\tau(v) = b^{-i}$ on average. The computational load and output rate may also be interpreted as node and edge weights, respectively. For $T_l(v)$ being the l-level sub-tree rooted in v, we extend the definitions to $\tau(T_l(v)) = \tau(v)$ and $\gamma(T_l(v)) = \sum_{u \in T_l(v)} \gamma(u)$. Note that $\gamma(T_l(v)) = l \cdot \gamma(v)$, because the accumulated rates of siblings equal the rate of the parent. For nodes u and v not in a common sub-tree, $\tau(\{u, v\}) = \tau(u) + \tau(v)$ and $\gamma(\{u, v\}) = \gamma(u) + \gamma(v)$. In particular, the computational load and output rate of any tree level equals 1. The *memory load* that a node v will place on the processor it is mapped to is a constant value c, because the node needs a fixed amount for buffering transferred data and for the internal data structures it uses for processing the data. For simplicity, one may assume $c = 1$ in the sequel.

We construct a mapping $\mu : V \to P$ of tree nodes to processors. Under this mapping μ, a processor P_i has *computational load*[1] $C_\mu(P_i) = \sum_{v \in \mu^{-1}(P_i)} \tau(v)$, i.e. the sum of

[1] The computational load depends on τ and thus averaged over time.

the load of all nodes mapped to it, and it has *memory load* $M_\mu(P_i) = \sum_{v \in \mu^{-1}(P_i)} c = c \cdot \#\mu^{-1}(P_i)$. The mapping μ shall have the following properties:

1. The maximum computational load $C^*_\mu = \max_{P_i \in P} C_\mu(P_i)$ among the processors shall be minimized. This requirement is obvious, because the lower the maximum computational load, the more evenly the load is distributed over the processors. With a completely balanced load, C^*_μ will be minimized.
2. The maximum memory load $M^*_\mu = \max_{P_i \in P} M_\mu(P_i)$ among the processors shall be minimized. The maximum memory load is proportional to the number of the buffers. As the memory per processor is fixed, the maximum memory load determines the buffer size. If the buffers are too small, communication performance will suffer.
3. As often as possible, sibling nodes shall be mapped to the same processor. We refer to the discussion on load balancing in Sect. 1.
4. The *communication load* $L_\mu = \sum_{(u,v) \in E, \mu(u) \neq \mu(v)} \tau(u)$, i.e. the sum of the edge weights between processors, shall be low.

Lemma 1 (Lower bounds). *In any mapping μ the maximum computational load is at least k/p, and the maximum memory load is at least $\lceil c \cdot (b^k - 1)/((b-1)p) \rceil$.*

We omit the routine proof of Lemma 1. The latter bound can be tightened for the case $p = k$. If no processor is overloaded, the root must be placed on a processor of its own, so that the rest of the tree is mapped onto $p - 1$ processors, leading to $M^*_\mu \geq c((b^k - 1)/(b - 1) - 1)/(k - 1) = c(b^k - b)/((b - 1)(k - 1))$.

For larger chip-multiprocessors, e.g. with $p \geq 20$, the assumption $k = p$ might lead to problems because the tree gets very large. In this case, we choose a small k, map the tree onto $p' = k$ *pseudo*-processors, and implement each pseudo-processor with p/k processors by evenly distributing the nodes assigned to that pseudo-processor. If fewer than p/k nodes are mapped to a processor (e.g. if the root is mapped separately), then we use a technique already known [4] and mentioned in [6]: we partition the very large data blocks and perform merges on the partitions in parallel.

2.2 ILP Formulation

In the following, we number the tree nodes in breadth-first order, i.e. the root gets index 1, its children 2, 3 etc., and generally, the ith child of an inner node v gets index $b \cdot (v - 1) + i + 1$, for $i = 1, 2, ..., b$. Let $V = \{1, ..., (b^k - 1)/(b - 1)\}$ denote the set of tree nodes, $V_{inner} = \{1, ..., (b^{k-1} - 1)/(b - 1)\}$ the set of inner nodes, and $P = \{1, ..., p\}$ the set of available SPEs. Our ILP formulation uses three arrays of $O(b^k \cdot p)$ boolean variables, x, y and z. The actual solution, i.e. the mapping of nodes to processors, will be given by x:

$$x_{v,q} = 1 \text{ iff tree node } v \text{ is mapped on processor } q.$$

In order to determine internal edges (where both source and target node are mapped to the same processor) and siblings on the same processor, we need to introduce auxiliary variables z and y:

$$z_{u,q} = 1 \text{ iff non-root node } u > 1 \text{ and its parent are mapped to processor } q.$$

$y_{u,q} = 1$ iff all children $b(u-1)+2, \ldots, b \cdot u + 1$ of inner node u are mapped to proc. q.

Also, we use an integer variable *maxMemoryLoad* that will indicate the maximum memory load assigned to any SPE in P, and integer variable *nSiblingsOnDiffSPEs* that will indicate the total number of inner nodes whose children are all mapped to the same processor. The following constraints must hold:
Each node must be mapped to exactly one processor, and each processor can be filled up to 100% with work[2]:

$$\forall v \in V: \quad \sum_{q \in P} x_{v,q} = 1 \qquad\qquad \forall q \in P: \quad \sum_{v \in V} x_{v,q} \cdot \tau(v) \leq 1$$

The memory load should be balanced:

$$\forall q \in P: \quad \sum_{v \in V} x_{v,q} \leq \mathit{maxMemoryLoad}$$

Communication cost occurs whenever an edge is not internal, i.e. its endpoints are mapped to different SPEs. To avoid products of two x variables when determining which edges are internal, we use the following constraints and slack variables z:

$$\forall v \in V_{inner},\ q \in P,\ i \in \{1,...,b\}: \quad \begin{aligned} z_{b(v-1)+i+1,q} &\leq x_{v,q} \\ z_{b(v-1)+i+1,q} &\leq x_{b(v-1)+i+1,q} \end{aligned}$$

and in order to enforce that a $z_{u,q}$ will be 1 wherever it could be, we have to take up the (weighted) sum over all z in the objective function. This means, of course, that only optimal solutions to the ILP are guaranteed to be correct with respect to minimizing memory load and communication cost.

The communication load is the total communication volume over all tree edges minus the volume over the internal edges:

$$\mathit{commLoad} = \sum_{v \in V - \{1\}} \tau(v) - \sum_{v \in V_{inner}} \sum_{q \in P} \left(\sum_{1 \leq i \leq b} z_{b(v-1)+i+1,q} \right) \cdot \tau(bv)$$

We apply the same trick to determine $y_{v,q}$:

$$\forall v \in V_{inner},\ q \in P,\ i \in \{1,...,b\}: \quad y_{v,q} \leq x_{b(v-1)+i+1,q}$$

The total number of nodes whose children are mapped to different processors is then

$$\mathit{nSiblingsOnDiffSPEs} \ = \sum_{v \in V_{inner}} \sum_{q \in P} (1 - y_{v,q})$$

Finally, the objective function is:

$$\text{Minimize} \quad \epsilon_M \cdot \mathit{maxMemoryLoad} \ + \ \epsilon_C \cdot \mathit{commLoad} \ + \ \epsilon_S \cdot \mathit{nSiblingsOnDiffSPEs}$$

where the positive weight parameters ϵ_M, ϵ_C and ϵ_S can be set appropriately to give preference to minimizing for *maxMemoryLoad*, *commLoad*, or *nSiblingsOnDiffSPEs* as first optimization goal. The formulation above requires that $\epsilon_C > 0$ and $\epsilon_S > 0$.

[2] We focus on the case $k = p$; the general case would need the constraint $\leq k/p$.

Table 1. The Pareto-optimal solutions found with ILP for $b = 2$, $k = p = 5,\ 6,\ 7$

k	5			6				7		
# binary var.s	305			750				1771		
# constraints	341			826				1906		
maxMemoryLoad	8	9	10	13	14	15	20	21	29	30
commLoad	2.5	2.375	1.75	2.625	2.4375	1.9375	1.875	2.375	2.3125	2.0

By choosing the ratio of ϵ_M to ϵ_C, we can only find two extremal Pareto-optimal solutions, one with least possible *maxMemoryLoad* and one with least possible *commLoad*. In order to enforce finding further Pareto-optimal solutions that may exist in between, one can use any fixed ratio ϵ_M/ϵ_C, e.g. at 1, and instead set a given minimum memory load to spend (which is integer) on optimizing for *commLoad* only:

$$\mathit{maxMemoryLoad} \geq \mathit{givenMinMemoryLoad}$$

2.3 ILP Optimization Results

We implemented the above ILP model in CPLEX 10.2 [8], a commercial ILP solver. Table 1 shows all Pareto-optimal solutions that CPLEX found for $b = 2$ and $k = p = 5, 6, 7$. The computations for $k = 5$ and $k = 6$ took just a few seconds each, the time to optimize for $k = 7$ varied between a few seconds and several hours per *givenMinMemoryLoad*. For $k = 8$, with 5088 binary variables and 6369 constraints, CPLEX exceeded the timeout of 24 hours and could only produce approximate solutions, including one with *maxMemoryLoad* of 37 and a *commLoad* of 2.78125, and one with 38 and 2.71875, respectively.

By varying ϵ_M/ϵ_C and keeping $\epsilon_S \ll \epsilon_C$, two of the Pareto-optimal solutions can be found, namely that with best *maxMemoryLoad* and that with best *commLoad*. As the memory load is often one order of magnitude larger than communication load, $\epsilon_C \gg \epsilon_M$ is necessary to spot the communication-optimal one. The remaining Pareto-optimal solutions in between can be found by setting *givenMinMemoryLoad* appropriately. We use a very small ϵ_S, to give the sibling placement optimization the least priority and not interfere with communication optimization. Figure 2 shows the generated tree drawings for two of the solutions for $k = 5$. The tree computed for $k = 7$ with minimum *commLoad* is shown in Figure 3.

2.4 A Divide-and-Conquer Based Approximation Algorithm

For larger values of k, we use the following divide-and-conquer algorithm (called DC-map in the sequel) which we first present for $b = 2$, and then extend to arbitrary b.

To construct a mapping for a k_1-level binary tree onto k_1 processors, we distinguish two cases. If $k_1 \leq k_0$, where k_0 is a constant, we take a precomputed optimal mapping. Currently we use $k_0 = 7$. If $k_1 > k_0$, we place the tree root onto one processor, and interpret the remaining $k_1 - 1$ processors as two sets of $k_1 - 1$ processors, each with half the computational power. We map a $(k_1 - 1)$-level tree onto each set recursively. Then

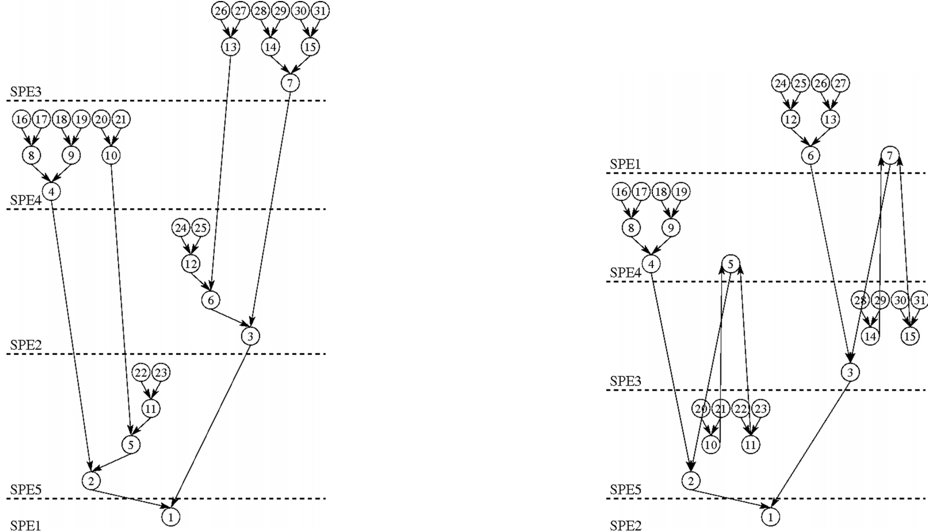


Fig. 2. Two Pareto-optimal solutions for mapping a 5-level tree onto 5 processors, computed by the ILP solver. — Left hand side: max. memory load 10 and communication load 1.75, obtained e.g. for $\epsilon_M = 0.1\epsilon_C$. — Right hand side: max. memory load 8 and communication load 2.5, obtained e.g. for $\epsilon_M = 10\epsilon_C$.

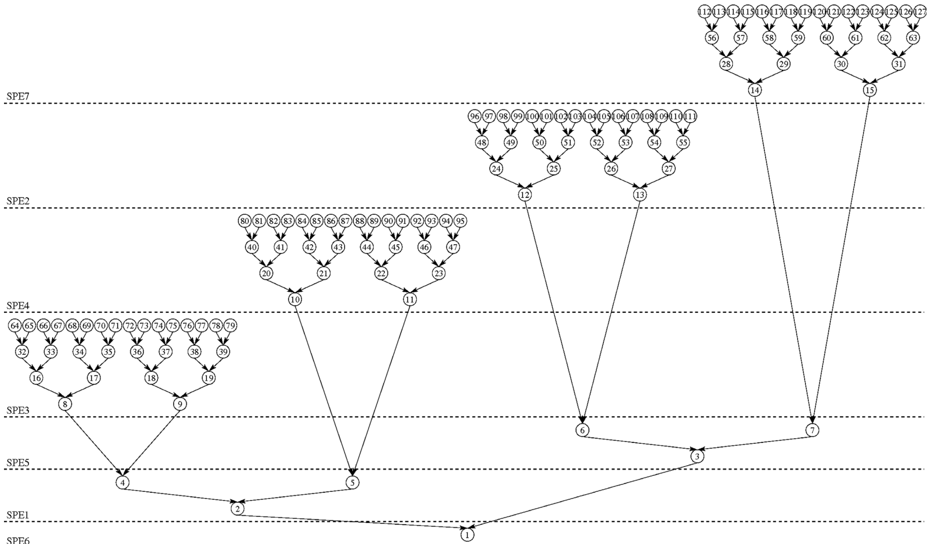


Fig. 3. The Pareto-optimal solution for mapping a 7-level tree onto 7 processors with least communication load, computed by ILP

we sort the processors in each set according to their memory load, one set in ascending order, one set in descending order. Finally we re-combine the i-th processors from both lists into one processor with full computational power.

We illustrate DC-map by an example where we employ an optimal mapping for $k_0 = 5$ (Fig. 2 right hand side), and construct a mapping for $k_1 = 6$. We first place the root of the 6-level tree onto processor 6. The two 5-level trees are mapped onto 5 'half'-processors each with the help of the optimal mapping, with memory loads of $8, 8, 7, 7, 1$. As this list is already sorted in descending order, we sort the copy in ascending order and receive $1, 7, 7, 8, 8$. Combination of the lists results in memory loads of $9, 15, 14, 15, 9$, and thus a maximum memory load of 15, compared to a sharpened lower bound of 13, but still representing a Pareto-optimal solution from Table 1.

To map a tree with $b > 2$, we receive b lists from the recursion step, that we may successively combine into pairs, as in a balanced binary tree with b leaves. Alternatively, we might use some form of linear optimization here.

DC-map mainly sorts lists of increasing length, thus its runtime is $\sum_{k=k_0}^{k_1} O(k \log k)$ = $O(k_1^2 \log k_1)$, which can be considered efficient given that typically $k_1 \ll 10^3$. By construction, DC-map produces a mapping where each processor has a computational load of 1. The maximum memory load may increase by a factor of b when going from k to $k+1$, because b lists are to be combined. In contrast, the lower bound increases by a factor

$$\frac{\frac{b^{k+1}-1}{(b-1)(k+1)}}{\frac{b^k-1}{(b-1)k}} \approx b \cdot \frac{k}{k+1}$$

Thus, if we start with an optimal solution for k_0 and use DC-map to construct a solution for $k_1 > k_0$, the maximum memory load may increase by a factor of $b^{k_1-k_0}$, while the lower bound increases by a factor $b^{k_1-k_0}k_0/k_1$. Thus, we may be away from the optimum maximum memory load by a factor of k_1/k_0.

DC-map does not take special care for the placement of siblings or communication load. Yet, with respect to siblings, the majority of the nodes and thus the siblings is in the levels close to the leaves, which are placed with the help of an optimal mapping. With respect to communication load, we may employ the following additional step. Normally, the two 'half'-processors to be combined into one are from different lists, i.e. they carry nodes from different subtrees that are not connected by edges. Yet, when the two lists are combined, we may interchange pairs of 'half'-processors with identical memory load without disturbing the algorithm. If the 'half'-processors to be interchanged are from different lists, then their partners in the combination are now from the same list, and the nodes they carry may be connected by edges that now become internal.

We have implemented a prototype version of DC-map, albeit without the improvement of communication load. We evaluated the prototype on the basis of optimal solutions for $k_0 = 3$ and $k_0 = 7$. Table 2 depicts the placement results achieved for $k_1 = 3, 4, \ldots, 8$ and $k_1 = 7, \ldots, 12$, respectively. From the numbers it is clear that the algorithm in practice is much closer to the lower bound than by a factor of k_1/k_0.

Table 2. Results for DC-map prototype

	$k_0 = 3$						$k_0 = 7$					
k_1	3	4	5	6	7	8	7	8	9	10	11	12
M_μ^*	3	6	8	15	24	46	21	42	84	132	236	453
lower bound	3	5	8	13	21	37	21	37	64	114	205	373
quotient	1.00	1.20	1.00	1.15	1.14	1.24	1.00	1.14	1.31	1.16	1.15	1.21
k_1/k_0	1.00	1.33	1.66	2.00	2.33	2.66	1.00	1.14	1.29	1.43	1.57	1.71

3 Sorting Algorithm and Performance

In order to test the usefulness of our mappings, e.g. with regard to load balancing, we implemented a discrete event simulation of the second phase of the parallel merge sorting algorithm. As the runtime of each merger node to produce one chunk of output is only dependent on the size of the output buffer, it is considered a constant. As furthermore communication and computation are assumed to be overlapped, we believe the simulation to quite accurately reflect the full algorithm. We chose $b = 2$ because quad-trees lead to high memory load, i.e. to very small buffer sizes, even for small k.

In each step, each SPE runs one merger node that has enough input data until it has produced one chunk, i.e. one output buffer full of data. As buffer size, we use 4 KByte for the output buffer (holding 1,024 32-bit integers), and 2×4 KByte for the input buffers, in order to allow a merger to commence work on its input data, while its input is being simultaneously filled with the output of a previous merger. Each merger mapped to a particular SPE receives a share of the SPE's processing time at least according to its position in the merge tree, i.e. a node at level $i \geq 0$ receives a share of at least 2^{-i}, provided that it has enough input to produce one chunk of output. We use a simple round robin scheduling policy in each SPE, where a merger receives a number of slots in proportion to its share. A merger not ready to run (e.g. insufficient input or full output buffer) is simply left out.

We have investigated three mappings resulting from our mapping algorithm. In the 5-level tree of Fig. 2 (right hand side), we have realized a 32-to-1 merge on 5 SPEs, with the restriction that no more than 8 mergers are to be mapped to one SPE. With 20 KByte of buffering (5 buffers of 4 KByte each) for each merger, this seems to be the upper limit. We used 32 input blocks of 2^{20} sorted integers each. The blocks were filled with randomly chosen integers and then sorted. The pipeline ran with an efficiency of 93%, meaning that in 93% of the time steps, the root merger node could run and produce output. In comparison to [6], our memory bandwidth requirements decreased by a factor of 2.5. Combined with a pipeline efficiency of 93%, we still gain a factor of 1.86.

By way of comparison, we also consider mapping a 4-level tree where leaf merger nodes over 4 SPEs, instead of 2, so that we use 6 SPEs in total. Thus, load balancing should not pose a problem. We have simulated this mapping with 16 input blocks of 2^{20} integers each, chosen as before. In all experiments, the pipeline ran with 100% efficiency as soon as it was filled. As we realize a 16-to-1 merge, we gain a factor of 2 on the memory bandwidth requirements in relation to [6]. Yet, as we need 6 SPEs instead of 4 (which would be the normal case $p = k$), our real improvement is only

$2 \cdot 4/6 = 4/3$ in this case. This mapping would be targeted towards a Cell BE variant with 6 SPEs as used in a Playstation 3.

Finally, we also simulated an 8-level tree on 8 processors. In this simulation, we had to reduce the buffer size to 256 bytes (64 integers), because the maximum memory load is 60 (resulting from DC-map with $k_0 = 2$), so that 300 buffers must be placed into the local memory of one SPE. As typically at most half the memory is available for data because of code size, and there are other data structures must be stored, too, using 75 KBytes for buffers seemed the upper limit. The simulation ran with an efficiency of at least 98% in all simulations. Thus, in comparison to the algorithm running with 4-to-1 mergers on 8 SPEs, i.e. on a complete Cell BE, our algorithm reduces memory bandwidth requirements by a factor of $0.98 \cdot \log_4(256) = 3.92$. Also we see that our algorithm can cope with rather small buffer sizes, as long as computation and communication can be overlapped.

4 Conclusion

We have investigated how to lower memory bandwidth requirements in the Cell BE by pipelining, with sorting being used as our case study. We have formulated the mapping of the merge tree onto the processors as an integer linear optimization problem, and given solutions for small tree sizes. For larger sizes, we presented a divide-and-conquer approximation algorithm. The mapping turns into a sorting algorithm whose performance we have demonstrated by a discrete event simulation. Note that our resulting sorting algorithm is also able to run on multiple Cell processors, as does [6]. At the beginning, there will be many blocks, and hence many b^k-to-1 mergers can be employed. In the end, when nearing the root, we are able to employ parallel mergers similar to the case $p > k$ discussed in Sect. 2.1. Approaches similar to the one presented here may work for other memory-intensive problems as well, such as data-parallel computations. We therefore plan to investigate other applications in the future.

References

1. Chen, T., Raghavan, R., Dale, J.N., Iwata, E.: Cell broadband engine architecture and its first implementation—a performance view. IBM J. Res. Devel. 51(5), 559–572 (2007)
2. Huh, J., Keckler, S.W., Burger, D.: Exploring the design space of future CMPs. In: Proc. Int.l Conf. Parallel Architectures and Compilation Techniques (PACT 2001), pp. 199–210 (2001)
3. Akl, S.G.: Parallel Sorting Algorithms. Academic Press, London (1985)
4. JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley, Reading (1992)
5. Gedik, B., Bordawekar, R., Yu, P.S.: Cellsort: High performance sorting on the cell processor. In: Proc. 33rd Intl. Conf. on Very Large Data Bases, pp. 1286–1207 (2007)
6. Inoue, H., Moriyama, T., Komatsu, H., Nakatani, T.: AA-sort: A new parallel sorting algorithm for multi-core SIMD processors. In: Proc. Int.l Conf. Parallel Architectures and Compilation Techniques (PACT 2007), pp. 189–198 (2007)
7. Shi, H., Schaeffer, J.: Parallel sorting by regular sampling. Journal of Parallel and Distributed Computing 14, 361–372 (1992)
8. ILOG Inc.: Cplex version 10.2 (2007), `http://www.ilog.com`

Towards an Intelligent Environment for Programming Multi-core Computing Systems

Sabri Pllana[1], Siegfried Benkner[1], Eduard Mehofer[1], Lasse Natvig[2], and Fatos Xhafa[3]

[1] University of Vienna, Department of Scientific Computing, Nordbergstrasse 15, 1090 Vienna, Austria
{pllana,sigi,mehofer}@par.univie.ac.at

[2] NTNU, Department of Computer and Information Science, Sem Saelands vei 9, NO-7491 Trondheim, Norway
Lasse.Natvig@idi.ntnu.no

[3] UPC, Department of Languages and Informatics Systems, C/Jordi Girona 1-3, 08034 Barcelona, Spain
fatos@lsi.upc.edu

Abstract. In this position paper we argue that an intelligent program development environment that proactively supports the user helps a mainstream programmer to overcome the difficulties of programming multi-core computing systems. We propose a programming environment based on intelligent software agents that enables users to work at a high level of abstraction while automating low-level implementation activities. The programming environment supports program composition in a model-driven development fashion using parallel building blocks and proactively assists the user during major phases of program development and performance tuning. We highlight the potential benefits of using such a programming environment with usage-scenarios. An experiment with a parallel building block on a Sun UltraSPARC T2 Plus processor shows how the system may assist the programmer in achieving performance improvements.

1 Introduction

While multi-core processors alleviate several problems that are related to single-core processors - known as *memory wall*, *power wall*, or *instruction-level parallelism wall* - they raise the issue of the *programmability wall*. On the one hand, program development for multi-core processors, especially for heterogeneous multi-core processors, is significantly more complex than for single-core processors. On the other hand, programmers have been traditionally trained for the development of sequential programs, and only a small percentage of them have experience with parallel programming.

Additionally, there is a portability problem. In the past programmers could trust that compilers succeeded to pass the increased computing power of next processor generations without high porting effort. This was due to relatively homogeneous processor designs even from different hardware vendors with instruction level parallelism (ILP) supported at hardware level. The architectural

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 141–151, 2009.
© Springer-Verlag Berlin Heidelberg 2009

change to multi-core processors, however, affects the programmer in several ways. On the one hand, thread level parallelism (TLP) must be exploited effectively and efficiently. In general, this cannot be done automatically by a compilation system, but requires assistance by the programmer. On the other hand, multi-core architectures differ significantly requiring that applications must be adapted to the various platforms.

While in the past only a relatively small group of programmers interested in HPC was concerned with the parallel programming issues, the situation has changed dramatically with the appearance of multi-core processors on commonly used computing systems. Traditionally parallel programs in HPC community have been developed by *heroic programmers*[1] using a simple text editor as programming environment, programming at a low-level of abstraction, and doing manual performance optimization. It is expected that with the pervasiveness of multi-core processors parallel programming will become mainstream, but it can not be expected that a mainstream programmer will like to become a HPC hero.

In this paper we argue that the programming productivity of multi-core[2] systems is increased if an intelligent programming environment would be available that (1) enables the programmer to work during the process of program development at a higher level of abstraction using domain-specific modeling languages in a model-driven development fashion; and (2) provides context-specific knowledge and performs iterative time-consuming tasks involved in program development in a semi automatic/autonomic manner (for instance, performance tuning). We propose a parallel programming methodology that combines model-driven and agent-supported program development with the use of high-level parallel building blocks. The goal is to increase programming productivity without restricting flexibility and creativity, allowing the programmer to fully use his/her intellectual capacity for software design at model-level. Although software development is considered to be an art, we anticipate that there are many implementation activities that can be performed more automatically/autonomically.

The rest of this paper is organized as follows. Section 2 describes our vision for programming of multi-core computing systems. We illustrate our approach experimentally in Section 3. Section 4 reviews the state-of-the-art in programming multi-core computing systems. We conclude the paper with a summary and future work in Section 5.

2 Intelligent Programming of Multi-core Systems

In this section we outline our methodology and the corresponding environment for programming multi-core systems.

---

[1] Andrea: "Unhappy is the land that breeds no hero." Galileo: "No, Andrea: Unhappy is the land that needs a hero." – Bertolt Brecht in *Life of Galileo*.

[2] Although some authors have introduced the term *many-core* to denote multi-core systems with many cores (i.e. 100 or more), we will stick to the more established term multi-core. We do not see a need to make a distinction between multi- and many.

2.1 Methodology

Our parallel programming methodology combines model-driven agent-supported program development with the use of high-level *parallel building blocks* (PBB). We propose to address the complexity of programming multi-core systems as follows:

- Raise the level of abstraction at which the programmer performs most of the activities during the process of software development, by using a model-driven development approach combined with PBBs;
- Support the programmer during the software development, by using intelligent software agents for providing context-specific knowledge and automation of iterative activities involved in software development and optimization.

Model-Driven Development (MDD) [13]. MDD is a software development method that advocates to *first model* a program and *then build* the program code. It is inspired by mature engineering disciplines such as *civil engineering*, where before an artifact (for instance a *bridge*) is built first the corresponding model is developed. In software engineering the models are usually described graphically using the Unified Modeling Language (UML). The model should preferably describe the program at an abstraction level that is independent from a specific platform. Models may be used to study the functionality and the performance of the program before the program code for a specific platform is developed. MDD has the potential to reduce software development time and complexity, by using tools for automatic model-to-code transformation and thereby reducing the programmer's effort for manual coding. Since multi-core architectures differ significantly from each other, a significant effort is required to adapt (that is *port*) programs to the various platforms. Since MDD captures the program logic as a platform-independent model, then program models remain largely unaffected from the changes in processor architectures. In our previous work we have developed an extension of UML for the domain of performance-oriented parallel/distributed programs [16] and the corresponding tool-support *Teuta* [10]. Teuta allows to build models of parallel programs, enrich them with performance-related information, and generate various textual representations (such as XML or C++).

Parallel Building Blocks. The PBBs are inspired from research in programming concepts such as *skeletons* [1,2,8] or *dwarfs* [3]. Basically, PBBs may be thought of as program-independent generic programming units that support software re-usability. A set of parameters is used to specify the functionality of a PBB in the context of a certain program. For instance, as parameter may serve the program-specific code (that is the code that PBB requires to perform the expected functionality in the context of a certain program). PBBs may be implemented for instance using *C++ Templates* or *Java Generics*. Parallelism is described within the PBB, and therefore the programmer is not exposed directly to the parallel programming complexity (such as dealing explicitly with the *communication and synchronization* among processing units or *deadlock avoidance*).

Commonly various combinations of PBBs may be used for solving a certain problem. In the context of programming environments PBBs lend themselves to an increased level of automation of various activities such as program transformation, code generation, performance optimization, and resource usage optimization. In our previous work, in the context of MALLBA project [1], we have developed a library of parallel skeletons (such as *branch and bound*, *metropolis*, *simulated annealing*, *genetic algorithms*, or *tabu search*) for solving various optimization problems.

Intelligent Software Agents. Software agents are programs that are *reactive*, *proactive*, *autonomic*, and *social* [21]. Software agents that have *learning* and *adapting* abilities are known as *intelligent software agents. Reactiveness* indicates the ability to respond adequately to changes in the context in which it operates. A *proactive* program performs activities to achieve a specific goal based on its initiative (it does not wait passively for a request of another entity to perform a certain activity). *Autonomy* indicates the ability to perform activities independently of user intervention in order to achieve a specific goal. *Social* programs are able to communicate and coordinate activities with other programs (that is agents). A program is considered intelligent if it is able to learn from the previous experience (for instance, via trial-and-error or generalization) and is able to adapt accordingly to the perceived changes in the environment. We have a vision about several intelligent software agents cooperating with each other and the programmer during the process of program development. Our vision is based on the idea that the programming environment should be better at helping the programmer as a more active partner. In our previous work, in the context of the AURORA project [4], we have used intelligent software agents to automate systematic performance analysis for parallel and distributed programs. Although software development is considered to be an art, we anticipate that there are many implementation activities that can be performed more automatically/autonomically using intelligent software agents.

In the following sub-section we propose a programming environment for multi-core computing systems that uses MDD, PBBs, and intelligent software agents.

2.2 Programming Environment

The proposed programming environment comprises a set of intelligent software agents that may help to automate the programming process at several levels. Some agents will advice the composition of programs using PBBs, while others will guide the exploration of different possible parallel strategies, load balancing and performance optimization (see Figure 1).

The programming environment provides the programmer with information feedback useful in the process of developing a program for a multi-core system. This information is collected at several levels, from program composition to information about resource usage (such as the cache behavior) obtained by execution or simulated execution. Also, information is exchanged between the agents at the system level in an automated manner continuously looking for ways of obtaining and improving knowledge about the performance of the program

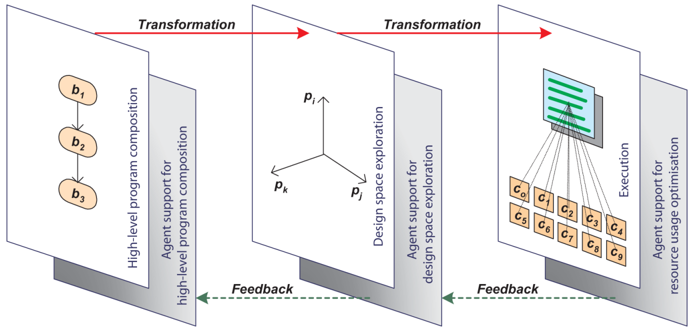


Fig. 1. Agent supported program development. The programming environment comprises multiple intelligent software agents that support program composition, design space exploration and resource usage optimization.

being developed. In this way, a parallel program with good performance can be developed with high programmer productivity.

In what follows in this section we highlight the major program development and tuning phases: (i) high-level program composition, (2) design space exploration, (3) resource usage optimization.

High-level Program Composition. This phase deals with the composition and coordination of PBBs. The granularity of PBBs may range from frequently used programming idioms, to larger patterns or dwarfs [3]. High-level descriptors are used to capture the main parallelization aspects of PBBs and serve as interface to agents in the design space exploration phase. The user composes the program graphically using a UML extension for multi-core systems.

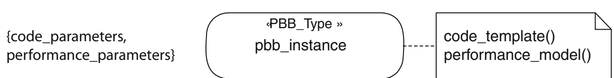


Fig. 2. UML representation of a PBB

The UML may be extended by defining new modeling elements, *stereotypes*, based on existing elements (also known as *base classes* or *metaclasses*). Stereotypes are notated by the stereotype name enclosed in guillemets `<<Stereotype Name>>`. Figure 2 depicts the graphical representation of a PBB. `<<PBB_Type>>` indicates the kind of PBB. With a PBB is associated the corresponding *parametrised code and performance model*. Parameters determine the behavior of the PBB instance in the context of a specific program.

The programming environment assists the user proactively during the program composition. For instance while the user is loading some old BLAS code for some dense linear algebra operations – the *composer agent* interrupts and suggests using the PBB for dense linear algebra tailored for efficient execution on

multi-core systems. Additionally, it may offer a list of other PBBs that often are used together with this one, as well as presenting typical compositional patterns in a graphical way.

Design Space Exploration. High level discrete-event simulation is used for rapid model-based performance evaluation of programs, using a hybrid method that combines mathematical modeling with high level discrete-event simulation [15].

For instance, after the completion of the program composition phase the programming environment may suggest to the user doing some high level rapid design space exploration. The estimated performance of various possible program implementations is presented by a *visualization agent*. While the user is studying the graphs, and gets some ideas for improvement, the programming environment is also analyzing the results and suggests changing some of the parameters in one of the PBBs (such as the parallelization granularity), and to perform some more detailed simulations for getting better knowledge of the performance that can be obtained with different task allocation and scheduling policies.

Resource Usage Optimization. Instruction-level simulation is used for more detailed studies of the utilization of shared resources such as shared on-chip memory and off-chip bandwidth. For instance, in [9] an efficient utilization of the shared cache resources has been found to have great affect on multi-core performance. This is integrated with the use of performance counters. A performance monitoring agent provides information about the state of the system (resource characteristics and usage). Instruction-level simulation is time consuming (may take several hours or days), and therefore should run in background. When finished, the findings will be propagated upwards back to the higher level performance models, as a model calibration process. It is a systematic way of bringing performance information from the execution (or simulated execution) environment back to the development environment. Please note that this kind of optimization is architecture-dependent.

For instance, the user may get hints from the programming environment for changes that will improve performance of the program. The programming environment may offer some detailed simulations at the instruction level, and helps the user to select those simulation experiments that are likely to be the most relevant. For instance, if higher-level simulations show that some of the processor cores were waiting for data for long periods, a more detailed study of the on-chip shared memory resources should be done.

3 Example

In this section we illustrate how best practices from HPC combined with agent based program development offer new opportunities to obtain efficient solutions.

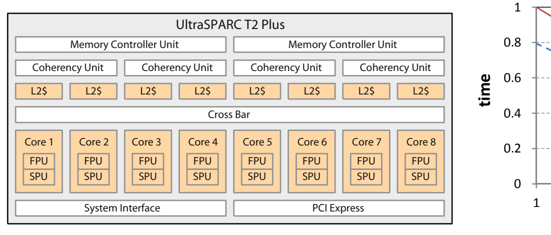
PBBs allow a programmer to specify various parallelization strategies together with the code and a first guess for individual parameters which are subject to the tuning process. This follows our assumption that only semi-automatic

parallelization is reasonable. The programmer specifies the main strategies for parallelizing the code and the system explores this restricted optimization space to generate efficient code. Two factors back up this approach. First, rich analysis work has been done in the past by the HPC community, including the authors institutions (Vienna Fortran Compilation System [6]), which can be reused. Second, in the past the strong emphasis on the target-code performance and manual performance tuning resulted in low programming productivity. The increasing importance of development of economically viable software nowadays reveals opportunities for semi-automatic parallelization, even at the price of achieving lower performance compared to a hand-tuned version.

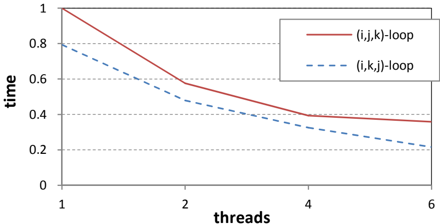
In our example we use as hardware platform the Sun UltraSPARC T2 Plus, codenamed Niagara-2, multi-core processor (shown in Figure 3(a)) which is an SMP extended version of the T2 allowing multiple Chip-level MultiThreading (CMT) processors to be used within a single system. The T2 Plus was presented in April 2008 and has up to 8 cores per processor with 8 hardware threads per core resulting in a maximum number of 64 threads per processor or logical CPUs as reported by the operating system. T2 Plus offers only poor support for instruction-level parallelism emphasizing thread-level parallelism. Two integer units are provided per core with four threads sharing one unit, and one FPU is provided per core with all eight threads sharing it. The L1 data cache has 8 KB per core and the on-chip L2 cache offers 4 MB which are shared between the cores.

In what follows in this section we present an example scenario to illustrate the agent-supported software development cycle. Different forms of PBBs are possible, but in the simplest case a PBB can be some loop nest together with data layout and work distribution annotations. Consider e.g. an application written in C consisting of a series of PBBs with one of them denoting a floating point matrix-matrix multiplication, i.e. `C[i,j] = C[i,j] + A[i,k] * B[k,j]` with loop nest (i,j,k). As parallelization strategy the programmer specifies that the elements of result matrix C should be assigned to processor cores in a row-wise manner and calculated by them. Since the target architecture is a Sun T2 Plus with 8 cores and 8 FPUs, the programmer specifies that the rows shall be assigned to 8 threads.

When submitted to the *design space exploration agent* and its analysis framework (cf. [6,7]), the framework detects poor spatial cache locality and performs



(a) Sun UltraSPARC T2 Plus.



(b) Performance improvements.

Fig. 3. Processor block diagram and optimization results

loop interchange resulting in loop nest (i,k,j). Then the code is split up in 8 threads as suggested by the programmer and assigned to the 8 cores of T2 Plus and executed. The monitoring component of the *resource usage agent* reveals low memory bandwidth utilization and low FPU utilization for this PBB and reports this feedback information to the agent. The *resource usage agent* is aware of the hardware characteristics of T2 Plus and knows about the hyper-threading (HT) technology provided by this kind of architecture with up to 8 hardware threads. Therefore the agent suggests to use HT technology to increase FPU utilization and reports to the *design space agent* to explore possibilities to increase the number of threads. Consequently, the *design space agent* proposes to assign the rows of result matrix C to 2, 4, 6 hardware threads per core resulting in a total number of 16, 32, 48 threads, respectively. Three versions are generated and submitted for execution. Moreover, feedback information is used by the compilation system to perform further optimizations (cf. [11]).

The key point is that this time-consuming tuning task is done automatically by the system and not by the programmer. The different versions are automatically generated and run on T2 Plus and the monitoring results are reported back to the agents and the programmer. Figure 3(b) shows the normalized execution times (longest execution time denoted by time unit 1.0) for the different versions with 1, 2, 4, 6 threads per core and the improvements achieved by the optimizations taking programmer annotations and hardware characteristics into account. The performance improvement of loop interchange is considerable and amounts to 26% for 1 thread per core, approx. 20% for 2 and 4 threads per core, and 66% for 6 threads per core. The performance improvement for increasing the number of threads per core to deal with memory latency is even more significant. The performance improvement assigning 2 and 4 threads to one core was for both loop nest versions approx. a factor of 1.7 and 2.5, respectively. For 6 threads per core we got for (i,j,k) loop nest a factor of 2.8 and for (i,k,j) loop nest up to 3.7. Based on this experience, the *resource usage agent* classifies increasing the number of threads to deal with memory latency as valuable optimization which has proven beneficial for this processor. The programming environment may suggest this kind of optimization for similar processor architectures as well.

4 A Review of the State-of-the-Art

An increasing number of research projects is addressing the challenge of programming multi-core computing systems. The *Habanero project* [12], which started in Fall 2007 at Rice University, aims to develop languages and compilers for the development of portable software for multi-core systems. The *SALSA project* [19] at Indiana University is investigating the use of services as building blocks for composing parallel data-mining applications based on the workflow paradigm. *Linked Sequential Activities* in SALSA, which are conceptually based on Communicating Sequential Processes of Hoare, are used to build services. The *Berkeley View* [3] project investigates the influence of multi-core processors in applications, hardware, programming models, and systems software for parallel computing. The

Berkeley View proposes to use a set of *dwarfs* (a dwarf defines a specific computation and communication pattern) for evaluation of parallel programming models. The recently established *Pervasive Parallelism Laboratory (PPL)* [17] at Stanford University is investigating future parallel computing platforms. PPL is supported by six computer and chip makers that are convinced that their product sales may decline if software is not able to use effectively the new multi-core-based hardware. *SWARM* [5], developed at Georgia Institute of Technology, is a parallel programming framework that provides a collection of primitives for programming multi-core processors. The *Programming Environments Laboratory (PELAB)* [18] at Linköping University is investigating the applicability of round-trip engineering techniques to parallelization of sequential programs. The *Cell Superscalar (CellSs)* [14] project at Barcelona Supercomputing Center focuses on parallelization of sequential programs for Cell BE processor. The CellSs parallelization involves the functional decomposition, code annotation and the use of a source-to-source compiler. The *IT Research Division of the NEC Laboratories Europe* [20] is investigating the use of work stealing concept to achieve load balancing.

In contrast to the related work we propose an intelligent programming environment that proactively supports the user during major phases of program development and performance tuning by providing context-specific knowledge and performing iterative time-consuming tasks involved in program development in a semi automatic/autonomic manner.

5 Conclusions

We have outlined an intelligent programming environment, which proactively supports the user during high-level program composition, design space exploration, and resource usage optimization. We have highlighted the potential benefits of using such a programming environment with usage-scenarios.

We have observed that even for a rather simple parallel building block such as matrix multiplication the exploration of the parameter space may be time prohibitive on one hand, but on the other hand there is a big potential for performance improvement. The example scenario described a first and manageable step towards an intelligent program environment for multi-core architectures. Several projects at the authors' home institutions are currently pursued towards the realization of such an intelligent programming environment for multi-core computing systems.

References

1. Alba, E., Almeida, F., Blesa, M., Cabeza, J., Cotta, C., Diaz, M., Dorta, I., Gabarro, J., Leon, C., Luna, J., Moreno, L., Pablos, C., Petit, J., Rojas, A., Xhafa, F.: MALLBA: A Library of Skeletons for Combinatorial Optimisation (Research Note). In: Monien, B., Feldmann, R.L. (eds.) Euro-Par 2002. LNCS, vol. 2400, p. 927. Springer, Heidelberg (2002)

2. Alind, M., Eriksson, M., Kessler, C.: BlockLib: A Skeleton Library for Cell Broadband Engine. In: International Workshop on Multicore Software Engineering (IWMSE 2008) at ICSE 2008, Leipzig, Germany, May 2008. ACM, New York (2008)
3. Asanovic, K., Bodik, R., Catanzaro, B., Gebis, J., Husbands, P., Keutzer, K., Patterson, D., Plishker, W., Shalf, J., Williams, S., Yelick, K.: The Landscape of Parallel Computing Research: A View from Berkeley. EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2006-183, December 18 (2006)
4. AURORA: A Priority Research Program on Advanced Models, Applications and Software Systems for High Performance Computing (1997–2007), `http://www.vcpc.univie.ac.at/aurora/`
5. Bader, D., Kanade, V., Madduri, K.: SWARM: A Parallel Programming Framework for Multi-Core Processors. In: First Workshop on Multithreaded Architectures and Applications (MTAAP) at IPDPS 2007, Long Beach, CA, USA, March 2007. IEEE, Los Alamitos (2007)
6. Benkner, S., Andel, S., Blasko, R., Brezany, P., Celic, A., Chapman, B., Egg, M., Fahringer, T., Hulman, J., Kelc, E., Mehofer, E., Moritsch, H., Paul, M., Sanjari, K., Sipkova, V., Velkov, B., Wender, B., Zima, H.: Vienna Fortran Compilation System - Version 1.2 - User's Guide. Technical report, Institute for Software Technology and Parallel Systems, University of Vienna (February 1996)
7. Benkner, S.: VFC: The Vienna Fortran Compiler. Scientific Programming 7(1), 67–81 (1999)
8. Cole, M.: Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. Parallel Computing 30(3), 389–406 (2004)
9. Dybdahl, H., Stenström, P., Natvig, L.: A cache-partitioning aware replacement policy for chip multiprocessors. In: Robert, Y., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2006. LNCS, vol. 4297, pp. 22–34. Springer, Heidelberg (2006)
10. Fahringer, T., Pllana, S., Testori, J.: Teuta: Tool Support for Performance Modeling of Distributed and Parallel Applications. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3038, pp. 456–463. Springer, Heidelberg (2004)
11. Gupta, R., Mehofer, E., Zhang, Y.: Profile Guided Code Optimizations. In: Srikant, Y.N., Shankar, P. (eds.) The Compiler Design Handbook: Optimizations & Machine Code Generation. CRC Press, Boca Raton (2002)
12. Habanero Multicore Software Project, `http://www.cs.rice.edu/~vs3/habanero/`
13. Model Driven Architecture, `http://www.omg.org/mda/`
14. Perez, J., Bellens, P., Badia, R., Labarta, J.: CellSs: Making it easier to program the Cell Broadband Engine processor. IBM Journal of Research and Development 51(5), 593–604 (2007)
15. Pllana, S., Benkner, S., Xhafa, F., Barolli, L.: Hybrid Performance Modeling and Prediction of Large-Scale Computing Systems. In: 2008 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008), Barcelona, Spain, March 2008. IEEE CS, Los Alamitos (2008)
16. Pllana, S., Fahringer, T.: On Customizing the UML for Modeling Performance-Oriented Applications. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, p. 259. Springer, Heidelberg (2002)
17. Pervasive Parallelism Laboratory, `http://ppl.stanford.edu/wiki/index.php/Pervasive_Parallelism_Laboratory`

18. Programming Environments Laboratory (PELAB), http://www.ida.liu.se/labs/pelab/
19. Service Aggregated Linked Sequential Activities (SALSA), http://www.infomall.org/multicore/
20. Wagner, J., Jahanpanah, A., Träff, J.: User-Land Work Stealing Schedulers: Towards a Standard. In: 2008 International Workshop on Multi-Core Computing Systems (MuCoCoS 2008) at CISIS 2008, Barcelona, Spain, March 2008. IEEE CS, Los Alamitos (2008)
21. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons Ltd., Chichester (2002)

Adaptive Read Validation in Time-Based Software Transactional Memory

Ehsan Atoofian, Amirali Baniasadi, and Yvonne Coady

University of Victoria
3800 Finnerty Road, Victoria BC V8P 5C2, Canada
{eatoofian,amirali}@ece.uvic.ca, ycoady@cs.uvic.ca

Abstract. In software transactional memory (STM) systems, read validation ensures that a transaction always has a consistent view of the memory. Existing read validation policies follow a static approach and use one policy across all applications. However, no single universal read validation policy offers optimal performance across all applications. We propose adaptive read validation (ARV) for time-based STMs and adjust read validation policy according to workloads' behavior. ARV not only varies read validation policy across applications, but also tunes read validation policy across different phases of a transaction. The adaptive nature of our suggested technique improves performance significantly for the set of workloads studied in this work.

Keywords: transactional memory, read validation policy, time-based transactional memory.

1 Introduction

Chip multiprocessors (CMPs) are becoming mainstream computing, making concurrent programming necessary to utilize available cores in CMPs. Traditionally, programmers use locks to develop parallel applications and to protect program critical sections from concurrent thread accesses. Coarse grained locks simplify programming; however, they are not scalable and work poorly when the number of threads increases. On the other hand, although fine-grained locks are scalable, they are complicated and error-prone. Priority inversion, deadlock, and other synchronization bugs make the lock-based programming too difficult for programmers, preventing developers from composing scalable applications out of existing software components.

The alternative solution for parallel programming is transactional memory (TM). Transactional memory alleviates problems associated with lock-based programming and enables developers to compose scalable applications. A transaction is a finite sequence of instructions that access memory and are executed by a thread. Each transaction is atomic: it commits if all read and write operations are validated, or aborts if it conflicts with any other transaction. The ability to abort and restart conflicting sections eliminates potential deadlocks and avoids dealing with the complexity of fine-grain locks. In addition to atomicity, transactions are linearizable [1]: they take effect in a one-at-a-time order. Linearizability allows transactions to run in isolation and prevents other transactions to interfere during execution. The underlying

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 152–162, 2009.
© Springer-Verlag Berlin Heidelberg 2009

system is free to reorder transactions but must ensure that the result of the execution is linearizable.

Transactional memory may be implemented in hardware (HTM) [2, 7], software (STM) [3, 6, 10, 16], or a combination of both [4, 18]. While HTM makes transactional memory fast, it increases design complexity and is not flexible. In addition, both HTM and hybrid approaches need new processor architectures. STM, however, can use available features of current processors and has fewer intrinsic limitations imposed by hardware structure, such as buffer size and caches.

In this work, we focus on STMs and introduce two adaptive read validation policies (ARV and ARV+) to improve performance in STMs. STMs use different policies to validate read operations and to guarantee a consistent view of memory. For example, in the eager validation policy [14], whenever a transaction reads a memory location, all previously read values are validated. Accordingly, the eager policy detects contention as soon as possible and enables a doomed transaction to abort instead of performing useless work. This, however, imposes an overhead which is quadratic function of the number of read operations. An alternative validation policy is the lazy policy [16] which postpones validation to the commit time. The lazy policy tends to minimize the window during which transactions may be identified as competitors; if application semantics allow both transactions to commit, lazy policy may result in significantly higher concurrency. However, lazy policy may waste resources on executing a doomed transaction. Our experiments show that applications react differently to eager and lazy policies. While some applications run faster using eager policy, others benefit more from the lazy policy. In adaptive read validation, we exploit this variability and use a speculative approach to adjust the validation policy based on applications' behavior. In the event that read operations of a transaction are likely to conflict, we use the eager policy to detect a conflict as soon as possible. On the other hand, if the history of a transaction shows that transactional reads usually commit without conflict, we select the less costly lazy policy.

The rest of the paper is organized as follows. In section 2, we explain the necessary background and discuss how a time-based STM works. Section 3 explains the intuition behind ARV and shows that neither eager nor lazy policies are the optimum policy across all applications. Section 4, discusses adaptive read validation in detail. We review related work in section 5. Finally, in section 6 we offer concluding remarks.

2 Background

In this section, we review time-based STMs. A time-based software transactional memory exploits global time to impose order among transactions and reason about consistency of data accessed by transactions. Time-based STMs eliminate the overhead of transactional memories which always verify consistency of memory for each transactional read [14]. Meanwhile, time-based STMs have a consistent view of memory at all times. This is in contrast to those STMs which postpone validation to commit time [16].

The global time in time-based STMs is implemented using a shared counter [3, 6, 9], and the counter is incremented by every committed write-transaction. A write-transaction is a transaction that stores at least one value to the memory. On the other

hand, a read-transaction only loads data from memory in the transactional section. Each transaction has a read-set and a write-set implemented using two separate link-lists. The read-set and write-set hold information for transactional reads and writes, respectively. We use TL2 [3] as a time-based STM for our evaluation framework.

In word-based STMs, e.g. TL2, each transactional write acquires a lock to prevent concurrent update of memory locations. TL2 exploits a shared array of locks, and a hash function maps memory address space to the array (Figure 1). Each array field has a size equal to the size of the address on the host machine. The least significant bit (lsb) of the lock shows the lock is free or acquired. If the lsb is zero (free), the rest of the lock shows the time stamp that the last transaction wrote to a memory address covered by the lock. If the lsb is one (acquired), the rest of the lock holds the address of the node in the write-set of the owner transaction. In both cases, the lock is word-aligned, and the lsb can safely represent the status of the lock (free or acquired).

At the start of each transaction, the global version clock is sampled and saved into a thread local variable called read version (*rv*). A transaction compares *rv* against the version number of locks corresponding to the transactional reads. If the version number is more than the *rv*, the memory location is updated after the transaction has started. Hence, the transaction is aborted; otherwise, a new node is allocated for the transactional read and is added to the read-set. At commit time, the nodes of the read-set are traversed to validate all transactional reads and detect any possible conflicts that may happen among concurrent transactions.

When storing to a memory location in a transactional section, the transaction allocates a new entry in the write-set. During commit time, all locks corresponding to the nodes in the write-set are acquired using bounded spinning to prevent deadlock [3]. Failure of acquisition results in an abort. After successful acquisition, the transaction increments the global version clock atomically using the CAS operation and saves it in the local variable write version (*wv*). Then, all elements in the read set are validated. This is necessary to satisfy atomicity of the transactional memory. If read validation fails, all acquired locks are released, and the transaction is aborted; otherwise, the write-set is traversed and memory locations are updated. Hence, TL2 follows write-back policy for transactional writes. Also, *wv* is written to the version section of all acquired locks, and lsbs of the locks are set to zero.

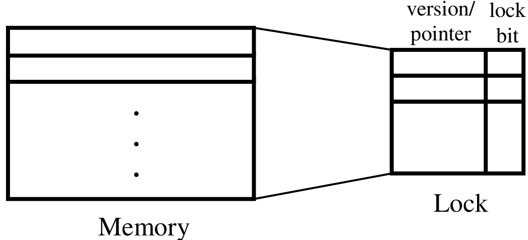


Fig. 1. Memory address space is mapped to an array of locks in word-based STMs. Each array entry shows whether the corresponding memory location is free or acquired [3].

It is important to note that in TL2 (and some other time-based STMs [9]), transactions validate the new read memory locations to keep a consistent view of the memory. For each transactional read, the version of the newly read memory location is compared against the local *rv*. Accordingly, read operations succeed only if their version is less than or equal to *rv*. In contrast to time-based STMs, other STMs incur significant overhead to have a consistent view of memory [5, 14, 17]. In the eager validation policy [5], transactions should validate all previously read elements for each new transactional load [5]. This is necessary as a memory location read by the current transaction may be written by other transactions later. This results in unsafe operations which may cause infinite loops, divide by zero, or other bugs. To alleviate the overhead associated with the eager policy, some STMs [10] exploit lazy validation with invisible reader policy [10]. In invisible reader policy, memory locations read by a transaction are not transparent to other transactions. As such, a memory location read by a transaction may be written by another transaction later, and the reader will not detect the conflict until the commit time. STMs with lazy validation and invisible reader policies do not prevent unsafe operations [10]. Instead, they exploit exception handling to catch a subset of inconsistencies and leave the rest to the programmers. This is not consistent with transactional memory's primary concern, i.e., simplifying parallel programming. One alternative is using lazy validation with visible reader policy [5]. In visible reader policy [5], write-transactions should check the list of readers to eliminate read-write conflict as soon as possible. This increases the complexity associated with transactional writes and results in heavy bookkeeping and cache miss penalties [15].

3 Motivation

In TL2, a transactional read is validated by checking that the corresponding memory location is updated before the start of the transactional section. Later, during the commit time, all memory addresses that have been read in the transactional section are validated to assure that they have not been written by other transactions since the last validation. This is similar to the lazy validation policy with one difference. In the lazy validation, the new read memory location is not verified. Verification of transactional reads is postponed to the commit time. However, as discussed in section 2, this increases complexity of programs which is too hard to deal with for most of programmers. We use term "semi-lazy" for the read validation method used in TL2.

While the semi-lazy policy reduces the conflict window size among transactions, it may waste significant computational resources on executing doomed transactions. A doomed transaction is a transaction that reads a memory location which later is written by another transaction. In the semi-lazy policy, the doomed transaction detects the conflict too late and at the commit time.

One solution to detect doomed transactions as early as possible is using eager validation. In the eager policy, whenever a transaction reads a memory location, it revalidates all previously read memory locations. As a result, a doomed transaction may be detected before commit time, and this may prevent wasting computational resources. However, validating all old read operations on each subsequent read is costly. For n read operations, the cost of read validation in the eager policy is $O(n^2)$.

We expect applications to react differently to semi-lazy and eager policies. In applications with frequent transactional conflicts, the eager policy may work better than the semi-lazy policy as quite often, a memory location read by a transaction is written by another transaction later. Hence, eager validation detects these conflicts early and prevents doomed transactions from wasting precious processor resources. On the other hand, in applications with low abort rate, the semi-lazy policy may result in better performance compared to the eager policy. This is due to the fact that in applications with less frequent conflicts, most of the transactions commit successfully. As such, an aggressive validation approach such as that used in the eager policy is not always necessary. Under such circumstances, performing the final validation at commit time could efficiently detect the few conflicting transactions.

To provide better understanding, in figure 2 we show the execution time for Stamp v0.9.7 benchmark suite [4] under the eager policy versus semi-lazy policy. Our experimental framework includes four 3.16 GHz dual-core Intel Xeon processors running Linux 2.4.21 (32-bit). Table 1 presents the set of benchmarks with their input arguments used in our experiments. We ran benchmarks up to completion and measured statistics over a set of ten test runs. Positive bars in figure 2 represent speed-up under the eager policy over the semi-lazy policy. For each benchmark, the number of threads varies from two to 64. This figure proves that neither eager nor semi-lazy policies are robust across all benchmarks. Moreover, within a single benchmark,

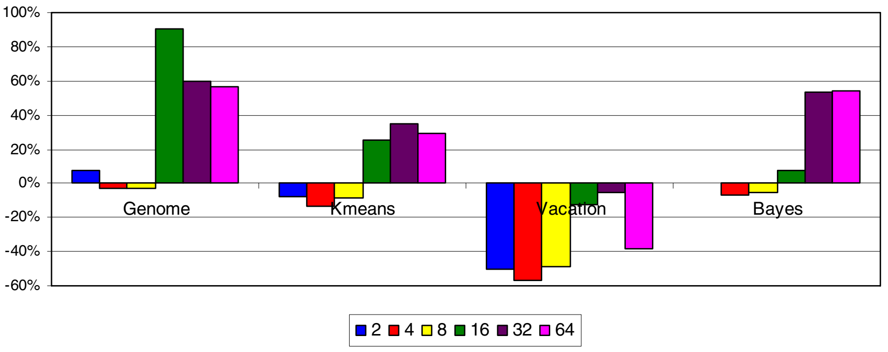


Fig. 2. Execution time for eager vs. lazy policy in Stamp v0.9.7 benchmarks. Positive bars show speed-up under eager policy. For each benchmark, the number of threads changes from two to 64.

Table 1. Stamp v0.9.7 benchmarks and input parameters

Benchmarks	Input Parameters
Genome	-g16384 -s64 -n4194304
Kmeans	-m20 -n20 -t0.00001 -i inputs/random-n65536-d32-c16.txt
Vacation	-n4 -q60 -u90 -r65535 -t4194304
Bayes	-v32 -r4096 -n10 -p40 -s1 -q1.0 -i2 -e8

depending on the number of threads, the optimum policy varies. For example, in *Genome*, when the number of threads is equal to four or eight, the semi-lazy policy outperforms the eager policy. However, when the number of threads is equal to two, 16, 32, or 64, the eager policy works better than the semi-lazy policy. In the next section, we exploit this variability and introduce adaptive techniques using one of the two policies according to the workload behavior.

4 Adaptive Read Validation

Adaptive read validation (ARV) takes a dynamic approach to select either the eager or the semi-lazy policy according to the applications' behavior. ARV relies on transaction history to make the proper decision. ARV stores the relative distance of the conflicting node in the read-set of a transaction. The relative distance is the distance of conflicting node from the head of link-list divided by the length of the link-list. If several read operations in the read-set fail, ARV selects the one appearing sooner in the link-list. The relative distance is compared with a predetermined threshold to select the validation policy for the subsequent read operations. If the relative distance is lower than the threshold, then eager policy is selected; otherwise, subsequent reads use the semi-lazy policy. The threshold is determined by running benchmarks with different thresholds and selecting the one with lowest execution time. Figure 3 explains ARV with an example.

ARV assumes that a transaction tends to repeat its behavior with regard to conflicts with other transactions. If a conflict happens early in a transaction (relative distance of conflicting node is less than the threshold), we would expect that the conflict would happen close to the starting point of the transaction next time the transaction executes too. Executing such a doomed transaction is waste of resources. Therefore, eager validation is the preferable policy as it detects the doomed transaction as soon as possible. On the other hand, if the relative distance of the conflicting node is larger than the threshold, ARV speculates that next time the transaction is executed, the conflict with other transactions will occur at a point closer to the end of transaction. Hence, ARV uses the less costly semi-lazy policy.

Figure 4 compares execution time under ARV to the execution time under semi-lazy (TL2 uses semi-lazy for read validation) in Stamp v0.9.7 benchmark suite [4].

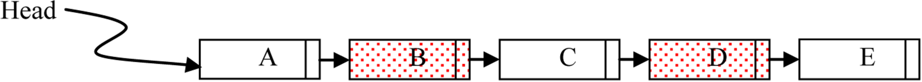


Fig. 3. The read-set has 5 nodes. In commit, the link-list is traversed to validate transactional reads. Assuming a threshold of 50%, if transactional read corresponding to node B conflicts, relative distance is 40% as the node B is the second node in a link-list with the length of 5. In this case, ARV uses eager policy for the subsequent reads as relative distance is lower than the threshold. If conflicting node is D, then the relative distance is 80%, and ARV sets the validation policy to semi-lazy.

The number of threads varies from two to 64 for each benchmark. In addition to the number of threads, we change threshold from five to 90. Positive bars represent speed-up for ARV. While in some benchmarks, e.g. *Genome*, ARV improves performance significantly, in some others, e.g. *Vacation*, ARV results in slowdown.

ARV uses the location of the conflicting node in the most recent read validation failure to speculate subsequent transactional read. While this technique may choose the optimum policy in benchmarks that are write-read conflict dominant, it may result in frequent misspeculations in benchmarks with a low write-read conflict rate. In transactions that write-read conflict occurs rarely, an invalid read may be followed by a long stream of successful transactional reads. In ARV, the optimum policy is selected by the last failed transactional read, and does not adjust the validation policy by the successful reads. This may result in considerable performance penalty. We clarify this issue through the following example.

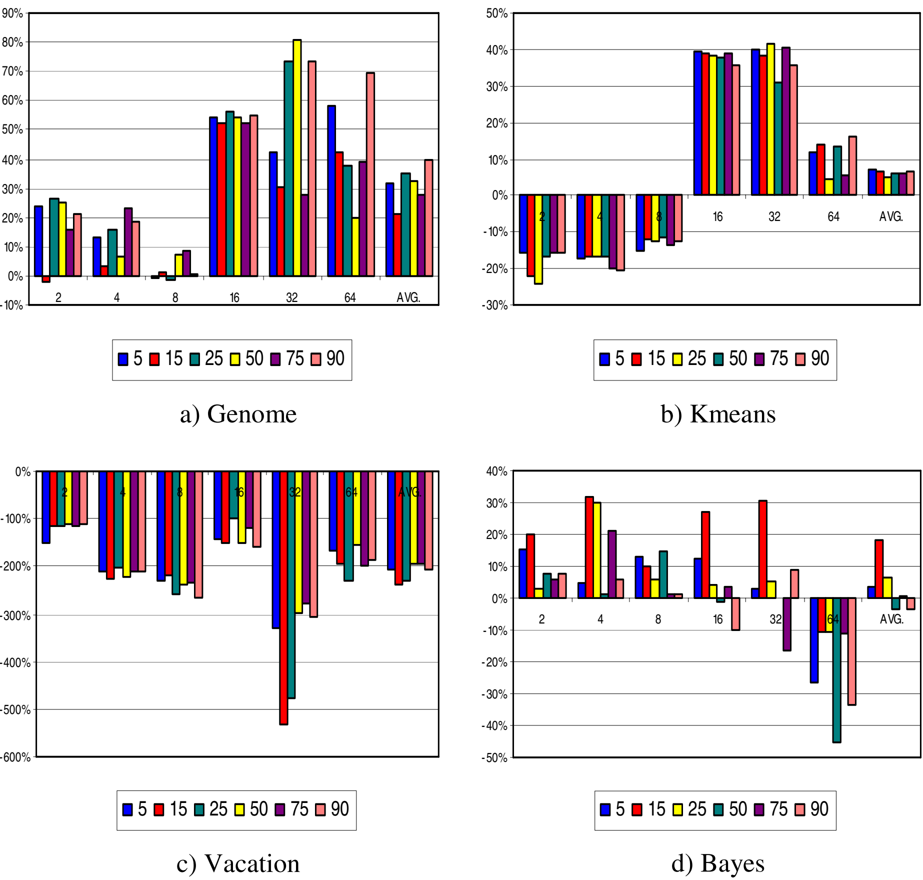


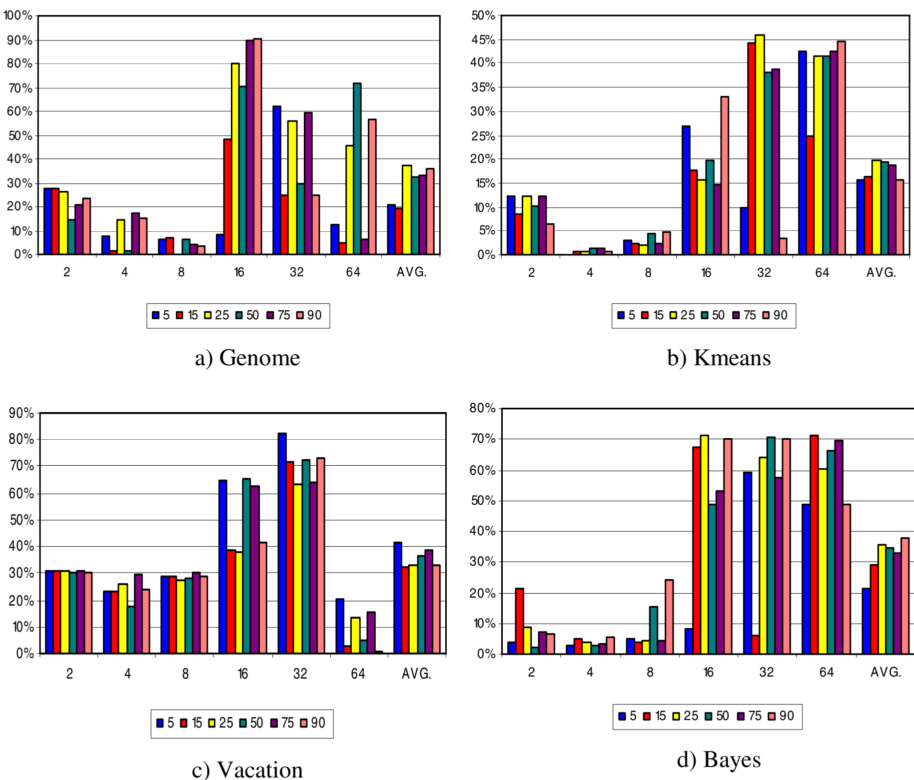
Fig. 4. Execution time in ARV relative to the semi-lazy policy for Stamp v0.9.7 benchmarks. Number of threads changes from two to 64, and threshold varies from five to 90.

Assume that transaction A has 10 transactional read operations, and the threshold in ARV is equal to 20%. When the transaction is executed for the first time, ARV picks the baseline read validation policy, i.e., semi-lazy. Assuming that a conflict is detected for the first transactional read at commit time, the relative distance of conflicting read will be 10% which is less than 20%. Therefore, ARV changes the validation policy to eager. Assuming a scenario where none of the transactional reads conflict after the transaction restarts, the selected eager policy will validate all previously read memory locations for each read operation imposing a significant timing overhead.

In the next section, we introduce ARV+ to dynamically change the validation policy for successful transactional reads.

4.1 ARV+

To adapt our validation policy in the event of successful reads, we use a predictor to speculate whether the next read would succeed or fail. If a read operation is predicted to succeed, ARV+ selects the semi-lazy for validation. However, if the read operation is likely to conflict, ARV+ picks the optimum policy taking into account the relative distance of the last conflicting read node and the threshold. As such, in the event of



a) Genome   b) Kmeans

c) Vacation   d) Bayes

Fig. 5. Execution time for ARV+ using 3-bit saturating counter relative to the semi-lazy policy for Stamp v0.9.7 benchmarks. The threshold for the saturating counter is six.

speculating successful reads, ARV follows the semi-lazy policy to validate read memory locations avoiding the timing overhead of the eager policy.

The predictor exploits saturating counters to speculate whether the subsequent read succeeds or fails. Saturating counters are widely used in hardware to speculate branch instructions [11] or values [12]. In our scheme, each transactional section has its own saturating counter. If a transactional read fails, the saturating counter is incremented. If the transactional read succeeds, the saturating counter is reset to zero. The subsequent read is predicted to succeed if the saturating counter is less than a static threshold; otherwise, it is predicted to fail. The value of an n-bit saturating counter changes from 0 to 2^n-1. The number of the bits and the counter threshold are configured once and used for all applications.

Figure 5 reports execution time for ARV+ and compares to the baseline TL2. We use 3-bit saturating counters with a threshold equal to six. We have picked these values after trying different configurations. ARV+ improves performance for all benchmarks. In some benchmarks, e.g. *Genome*, ARV+ improves performance up to 90%. By exploiting saturating counters, ARV+ is able to predict those transactional reads that pass validation and select the semi-lazy policy to validate those reads. However, without the saturating counter, ARV+ could mistakenly select the eager policy for such transactional reads and incur significant timing overhead.

5 Related Work

Transactional memory was originally proposed as a hardware technique by Herlihy and Moss [2] and later by Stone et al. [7]. Shavit and Touitou [8] introduced the first transactional memory relying on software. Previous studies proposed two types of STMS: word-based [3, 13] and object-based [14]. Word-based STMs access memory at granularity of memory words or larger blocks. Object-based STMs access memory at the granularity of objects. While object-based STMs need compiler or manual insertion of lock fields, word-based STMs do not modify data structures.

TL2 [3] uses an integer counter which is shared among all threads as a time base. The overhead of this counter is acceptable as inter-processor communication is done through on-chip wires, and so sharing data in current chip multiprocessors is inexpensive. TL2 uses invisible reads with semi-lazy policy to guarantee consistency. This simplifies read validation as TL2 only checks the current read operation. However, such a read validation policy does not necessarily prevent a doomed transaction to abort before commit time. We use TL2 as our experimental framework and propose ARV+ to detect a doomed transaction as soon as possible and abort before commit.

Herlihy et al. [14] exploit the eager policy in DSTM to validate transactional reads. DSTM avoids potential inconsistency by maintaining a private read list (invisible reader) that remembers all values previously returned by transactional reads. On every subsequent read, the transaction checks the validity of these values and aborts if any of the reads is not valid. Invisible reader incurs significant validation overhead which is quadratic function of transactional reads. Visible readers solve the problem. A writer transaction explicitly aborts all visible readers. As such, the quadratic overhead of eager policy is eliminated. Unfortunately, visible readers achieve this improvement at the expense of a significant increase in bookkeeping and cache eviction [15].

ARV+ is different from DSTM as it pays the overhead of aggressive validation only when there is high confidence that the subsequent read conflicts occur early in the transactional section without side effect on bookkeeping and cache miss.

In contrast to DSTM, OSTM [16] uses the lazy validation policy and checks transactional reads at commit time. Using the lazy policy with invisible reads in OSTM allows a transaction to enter an inconsistent state during its execution. Inconsistency may cause memory access violations, infinite loops, or other faults in programs. Fraser proposes a mechanism based on exception handling [16] to catch problems when they arise. On a memory access violation, the exception handler aborts the transaction that caused the exception. The responsibility of detecting other inconsistencies is left to the application programmer which makes programming very difficult. ARV+ does not impose such complexities on programmers.

Marathe et al. [17] proposed ASTM which uses an adaptive technique for object acquisition. ASTM switches from DSTM-style eager acquire [14] to OSTM-style lazy acquire [16] based on the number of transactional writes. ARV+ is different as it changes the read validation policy and not the object acquisition policy. In addition, our speculative technique is different, and it relies on relative distance and saturating counters.

6 Conclusion

In this work, we evaluated eager and lazy validation policies and showed that there is no single validation policy offering optimal performance across all applications. To address this inefficiency, we introduced a new read validation policy that adapts based on workload characteristics. We proposed ARV which selects one of the two validation policies using the relative distance of the conflicting node. Moreover, we introduced ARV+ to improve performance further. ARV+ exploits saturating counters to speculate successful transactional reads and is able to improve performance for Stamp v0.9.7 benchmarks up to 90%.

References

1. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. Trans. on Prog. Lang. and Syst. 12, 3 (1990)
2. Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: Proceedings of ISCA (1993)
3. Dice, D., Shalev, O., Shavit, N.: Transactional locking II. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 194–208. Springer, Heidelberg (2006)
4. Minh, C.C., Trautmann, M., Chung, J., McDonald, A., Bronson, N., Casper, J., Kozyrakis, C., Olukotun, K.: An Effective Hybrid Transactional Memory System with Strong Isolation Guarantees. In: Proceedings of ISCA (June 2007)
5. Marathe, V.J., Scherer, W.N., Scott, M.L.: Design tradeoffs in modern software transactional memory systems. In: Proceedings of the 7th Workshop on Languages, Compilers,and Run-time Support for Scalable Systems, LCR 2004 (2004)
6. Saha, B., Adl-Tabatabai, A.-R., Hudson, R.L., Minh, C.C., Hertzberg, B.: McRT-STM: a high performance software transactional memory system for a multi-core runtime. In: Proceedings of PPOPP, March 2006, pp. 187–197 (2006)

7. Stone, J.M., Stone, H.S., Heidelberger, P., Turek, J.: Multiple reservations and the Oklahoma update. IEEE Parallel & Distributed Technology: Systems & Technology, 58–71 (November 1993)
8. Shavit, N., Touitou, D.: Software transactional memory. In: Proceedings of PODC (August 1995)
9. Riegel, T., Fetzer, C., Felber, P.: Snapshot Isolation for Software Transactional Memory. In: 1st ACM SIGPLAN Workshop on Transactional Computing (June 2006)
10. Fraser, K.: Practical Lock-Freedom. Technical Report UCAM-CL-TR-579, Cambridge University Computer Laboratory (February 2004)
11. Tse-Yuh, Y., Patt, Y.: Alternative implementations of two-level adaptive branch prediction. In: Proceedings of the 19th Annual International Symposium on Computer Architecture (May 1992)
12. Lipasti, M.H., Wilkerson, C.B., Shen, J.P.: Value locality and load value prediction. In: ASPLOS, October 1996, pp. 138–147 (1996)
13. Harris, T., Fraser, K.: Language support for lightweight transactions. In: Proceedings of OOPSLA, October 2003, pp. 388–402 (2003)
14. Herlihy, M., Luchangco, V., Moir, M., Scherer III, W.: Software transactional memory for dynamic-sized data structures. In: Proceedings of PODC, July 2003, pp. 92–101 (2003)
15. Spear, M.F., Marathe, V.J., Scherer III, W.N., Scott, M.L.: Conflict Detection and Validation Strategies for Software Transactional Memory. In: Proc. of DISC, Stockholm, Sweden (September 2006)
16. Fraser, K.: Practical Lock-Freedom. Ph.D. dissertation, UCAMCL-TR-579, Computer Laboratory, University of Cambridge (Feburary 2004)
17. Marathe, V.J., Scherer III, W.N., Scott, M.L.: Adaptive software transactional memory. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 354–368. Springer, Heidelberg (2005)
18. Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M., Nussbaum, D.: Hybrid transactional memory. In: The Proceedings of the 12th Intl. Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA (October 2006)

Compile-Time and Run-Time Issues in an Auto-Parallelisation System for the Cell BE Processor

Alastair F. Donaldson[1], Paul Keir[2], and Anton Lokhmotov[3]

[1] Codeplay Software, 45 York Place, Edinburgh, EH1 3HP, UK
[2] Department of Computing Science, University of Glasgow, 18 Lilybank Gardens, Glasgow, G12 8QQ, UK
[3] Department of Computing, Imperial College London, 180 Queen's Gate, London, SW7 2AZ, UK

Abstract. We describe compiler and run-time optimisations for effective auto-parallelisation of C++ programs on the Cell BE architecture. Auto-parallelisation is made easier by annotating *sieve scopes*, which abstract the "read in, compute in parallel, write out" processing paradigm. We show that the semantics of sieve scopes enables data movement optimisations, such as re-organising global memory reads to minimise DMA transfers and streaming reads from uniformly accessed arrays. We also describe run-time optimisations for committing side-effects to main memory. We provide experimental results showing the benefits of our optimisations, and compare the Sieve-Cell system with IBM's OpenMP implementation for Cell.

1 Introduction

The Cell Broadband Engine (BE) processor [4] is a heterogeneous multi-core chip, which consists of a Power Processing Element (PPE) and eight Synergistic Processing Elements (SPEs). To avoid memory bottlenecks, each SPE is equipped with 256KB of fast local memory, which can be viewed as an extended register file for intensive calculations, and accesses main memory via DMA transfers. This approach allows scalable parallelisation over SPEs for suitable algorithms. Abandoning the convenient shared memory paradigm, however, makes the Cell processor difficult to program correctly and efficiently: the programmer needs to write separate programs for the PPE and SPEs, pack data into vectors for SIMD processing, and orchestrate data movement explicitly using untyped DMA transfers.

Codeplay's Sieve C++ [2,5] is a C++ extension to aid automatic parallelisation. The principal language construct is the *sieve block* – a lexical scope prefixed with the **`sieve`** keyword. By placing code inside a sieve block, the programmer instructs the compiler to *delay* writes to memory locations defined outside the block (global memory) and apply them *in order* on exit from the block. Conceptually, global memory is read on entry to the block and written to on exit from the block. Thus, the compiler is free to re-order computation within a sieve block, if it has no dependences on memory locations defined within the block (local memory). Restricting dependence analysis to local memory makes C++ code more amenable to deterministic automatic parallelisation.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 163–173, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In the context of the Cell processor, a sieve block makes explicit the notion of separate memory spaces: code outside a sieve block runs on the PPE and accesses main memory as usual; code inside a sieve block is a candidate for parallelisation over SPEs, with local variables to be placed in local store. The sieve semantics enables streaming between global memory and local store.

Sieve blocks are similar to *tasks* in Stanford's Sequoia [3] and BSC's CellSs [1] in that they specify a fragment of code to be executed on SPEs. Unlike a task, a sieve block leaves unspecified the working set of code, making optimisation of sieved code more challenging. However, while a task (a leaf task in Sequoia) is intended for execution on a single SPE, the sieve semantics parallelisation of a sieve block across multiple SPEs. The sieve construct is similar to the bulk-synchronous parallel (BSP) model [6]: it separates computation (on data brought into local memory) and communication (of results into global memory). Communication in BSP, however, is non-deterministic.

We have previously described Sieve C++ and its other constructs facilitating automatic parallelisation via software thread-level speculation [2]. Our contributions in this paper are: a discussion of the components of the Sieve-Cell system (§2); the description of optimisation techniques concerning the movement of data between main memory and local store (§3); and a comprehensive experimental evaluation showing the speedups afforded by our optimisations and comparing the Sieve-Cell system with IBM's OpenMP implementation for Cell (§4). We conclude (§5) with an outline of future work.

2 Sieve Overview

We illustrate the sieve concept and its advantages using a molecular dynamics example (§2.1), and describe a sieve implementation for the Cell BE processor (§2.2).

2.1 Sieve Scopes and Outer Pointers in a Molecular Dynamics Example

In addition to marking sieve blocks, the **sieve** keyword can be used as a function qualifier indicating that the function may be called from a sieve block (or other sieve functions) and therefore should be compiled with the semantics of delayed writes to global memory. (Sieve blocks and functions constitute sieve *scopes*.)

The **outer** qualifier applied to a pointer declared within a sieve scope indicates that the pointer points to data in global memory. (Pointers declared outside a sieve scope are outer by default.) Hence, writes via outer pointers occuring in a sieve scope get delayed.

The following listing shows a function, `computeForces`, which takes input arrays representing masses and positions for each particle in a system, and computes an output array representing forces exerted upon each particle, according to the law of gravity:

```
extern int Size;
sieve float3 rNormalised(outer float3 *Pos, int i, int j);

*Pos, float* Mass) {
  sieve {
    for(int i=0; i<Size; ++i) {
      float3 Potential = { 0.0f, 0.0f, 0.0f };
```

```
      for(int j=0; j<Size; ++j)
        Potential -= rNormalised(Pos, i, j) * Mass[j];
      Forces[i] = Potential * Mass[i]; // Delayed write
  } } // Side-effects to Forces[] committed here
}
```

Listing 1. Molecular dynamics code, annotated with a sieve block

We do not show code for the `float3` class (a floating point vector class with standard operations), or for the sieve function `rNormalised` which, given `Pos`, `i` and `j`, returns $(0,0,0)$ if `i`=`j`, and $(\texttt{Pos[i]}-\texttt{Pos[j]})/|(\texttt{Pos[i]}-\texttt{Pos[j]})|^3$ otherwise.

Consider the standard C++ code obtained by removing the **`sieve`** and **`outer`** keywords from Listing 1. In this form, the code is hard to automatically parallelise for the following reasons. First, the compiler must conservatively assume that the arrays `Forces` and `Mass` may overlap, which would lead to a carried dependence on the outer loop due to the write to `Forces`. Second, note that both loops are bounded by the external global variable `Size`. The compiler has to assume that `Size` may be modified during calls to the `rNormalised` function (for which the compiler does not necessarily have source code), which would destroy the regular structure of the loops. Similarly, the array `Pos` is passed as a parameter to `rNormalised`. The compiler must therefore assume that memory accessed via `Pos` could be modified by this function call, which would introduce carried dependences on both loops in the nest.

For the molecular dynamics example, this conservativeness is due to the mechanical compiler lacking domain-specific information. In a sensibly-written application, `computeForces` will be called with arrays that do *not* overlap, and `Size` and `Pos` will *not* be modified by the pure function `rNormalised`. The intention that the input parameters refer to distinct memory regions can be stated using the C99 **`restrict`** qualifier, but restricted pointers do not help with a possible modification by the called function.

The sieve block of Listing 1 tears down these barriers to parallelisation. The `Pos`, `Mass` and `Forces` parameters are outer pointers, since they are declared outside the sieve block. As a result, during sieve block execution, any modification to data via these pointers is delayed until the end of the block. This change in semantics means that there are no loop-carried dependences even if the arrays do overlap. The compiler knows that `rNormalised` is a sieve function, compiled with delayed semantics, so that if the function (or a function it calls in turn) writes to global variable `Size`, or via outer pointer `Pos`, these writes will be delayed, having no effect on sieve block computation.

Clearly, the semantics of the sieve block will depart from the conventional semantics if the code does involve a write to, followed by a read from, a global memory location. While parallelising such code with standard semantics would result in non-deterministic, undefined behaviour, the sieve semantics mean that parallel code is deterministic, regardless of how many cores are employed. If sieved code does not behave as expected (due to a programmer error, or to a sieve block being applied to code which involves read-after-write dependences by design) the guarantee of determinism means that the programmer can debug their multi-core application on a *single* core.

2.2 The Sieve-Cell System

The Sieve concept fits neatly with systems having multiple levels of memory hierarchy, in particular, the Cell BE processor. The programmer uses a sieve block to specify that a portion of code should be distributed across the SPEs. Variables declared inside sieve scopes reside in SPE local store; variables declared in standard scopes are located in main memory, and data structures in main memory can be traversed on an SPE via outer pointers. A read inside a sieve block from global memory results in the transfer of data into local store via DMA; a write to global memory results in an entry being appended to a queue of side-effects, to be applied at the end of the sieve block. The Sieve compiler and run-time system take care of the low-level details associated with data movement. The programmer is able to write a unified application for the Cell processor, rather than being forced to write separate PPE and SPE programs with explicit communication via mailbox messages and DMA transfers (which cannot be typechecked). Annotating blocks of code, as opposed to outlining code into functions, also aids productivity, allowing Sieve versions of serial C++ codes to be developed quickly.

Sieve Compiler. The compiler processes a Sieve C++ application and outputs a set of ANSI C files, which we refer to as OutputC files, together with a makefile. The makefile uses third party C compilers for the PPE and SPE processors to compile these C files, linking the resulting object files with the Sieve run-time library and other PPE/SPE libraries to produce a Cell executable. The Sieve compiler provides full support for PPU and SPU vector intrinsics.

The **`sieve`** and **`outer`** keywords allow type-checking across the PPE/SPE boundary, ensuring for example that SPE code does not accidentally de-reference a pointer to PPE data. The occurrence of a sieve block in a Sieve C++ application causes a call to a function named `runSieve` to be generated at a corresponding position in the OutputC code. This function is part of the Sieve-Cell PPE run-time library. Given a pointer to an SPE program for the associated sieve block, the `runSieve` function manages the distribution of sieve scope execution across available SPEs.

PPE Runtime. After loading each SPE with the sieve block program,[1] the PPE run-time issues each SPE with a speculative work unit. A work unit consists of a program point in the sieve block at which execution should begin, called a *split point*, together with speculated values for variables which are live at the given split point, and an integer specifying how many further split points should be crossed before execution of the work unit is completed [2]. Execution of a work unit also completes if the end of the sieve block is reached before the given number of split points is crossed. To ease speculation (*e.g.* for automatic loop parallelisation), the compiler requires that the only variables live across split points are *iterators* – instances of special user-defined classes having methods for prediction of class state after traversal of a given number of split points.

After issuing work units, the PPE run-time sleeps, waking up when an SPU thread interrupts to indicate completion of work. On receiving a completed work unit from an SPE, the PPE issues the SPE with a fresh work unit, then attempts to *validate* the completed work – checking that the predicted iterator values for the work unit match

[1] A run-time check ensures that SPEs are not needlessly re-issued sieve block code if the same sieve block is executed in succession.

the actual values computed by the previous work unit. A completed work unit contains a queue of associated side-effects. Side-effects for a valid work unit are kept until the end of the sieve block; side-effects for an invalid work unit are discarded. At the end of the sieve block, the PPE run-time invokes a function, `runSideEffects`, which takes the side-effects generated by all valid work units (by which time all side-effects have been transferred to main memory), and commits these side-effects *in order*.

SPE Runtime. The SPE program for a sieve block consists of a small run-time system loop together with code for work unit execution, which must all fit within the SPE local store. Each SPE has an outbound interrupt mailbox, which it uses to request a work unit from the PPE. Using the interrupt mailbox means that the PPE can sleep when not servicing an SPE, avoiding a busy-wait loop. The PPE responds to the SPE, via the SPE's inbound mailbox, with a pointer to a work unit. The SPE uses this pointer to fetch the work unit via DMA, after which the SPE executes the work unit.

By default, reads from main memory are via an SPE software cache, based on an implementation provided by IBM as part of the Cell SDK for Linux. Alternative methods for reading data from main memory are discussed in §3. For each write to main memory inside a sieve scope, the compiler generates a call to the `delayedStore` function in the SPU run-time. This function takes pointers to a SPU source address and PPU destination address, and an integer specifying the size of the data to be stored. The function adds a (*PPU address*, *size*, *data*) triple to a queue on the SPE. When an SPE's local side-effect queue reaches an upper bound (specified at compile-time), the SPE transfers the contents of its queue to a temporary location in main memory, and continues execution with an empty local queue. The PPE run-time is responsible for managing SPE requests to allocate main memory for temporarily storing side-effects.

3 Sieve-Cell Optimisations

Efficient data movement is key to achieving high-performance on the Cell processor. We describe compiler and run-time data-movement optimisations in the Sieve-Cell system.

3.1 Streaming DMA Reads

A common DMA transfer optimisation involves fetching data from main memory in large chunks before processing. For example, given an SPU loop which on each iteration fetches and processes an element of main memory array `A[0..N-1]`, it is typically more efficient to transfer `A[0..N-1]` into local memory *before* executing the loop. This is due to high cost of initiating a DMA transfer compared with the cost of transferring additional bytes once a DMA has been initiated. If `N` is large then it may be worth overlapping communication with computation, streaming data from `A` in chunks and using double-buffering to process chunk n while fetching chunk $n+1$. Indeed, streaming may be necessary if `N` is large enough that `A[0..N-1]` does not fit into local store.

The Sieve compiler exploits the delayed semantics to generate efficient DMA streaming code when compiling regularly structured loops which read through **`outer`** pointers. We illustrate this using the molecular dynamics example of Listing 1 as follows:

```
void computeForces(float3 *Forces, float3 *Pos, float* Mass) {
  sieve {
    DMAStream<sizeof(float)> MassStr_i, MassStr_j;
    int LocalSize = Size;
    MassStr_i.start(Mass);
    for(int i=0; i<LocalSize; ++i) {
      float3 Potential = { 0.0f, 0.0f, 0.0f };
      MassStr_j.start(Mass);
      for(int j=0; j<LocalSize; ++j)
        Potential -= MassStr_j.read(j) * rNormalised(Pos, i, j);
      Forces[i] = Potential * MassStr_i.read(i);
    }
    MassStr_j.destroy(); MassStr_i.destory();
} }
```

Listing 2. Optimised molecular dynamics code

The compiler spots that the outer loop of the sieve block includes a statement reading from the base address `Mass` with offset `i`. Since `i` can be identified as an index variable for the outer loop, the compiler generates a DMA stream object, `MassStr_i`, and replaces the read from `Mass[i]` with a read from `MassStr_i`. Similarly, the regular reads from `Mass` offset by index variable `j` in the inner loop are replaced with reads from a stream, `MassStr_j`.[2]

A DMA stream can be thought of as a window into an array in main memory. In our double-buffered implementation, a stream is an SPU-side record consisting of a pair of buffers, a pointer to the *current* buffer, a base address for the PPU array to which the stream corresponds, an address indicating the main memory address to which the current buffer refers, and a DMA tag to monitor completion of prefetching operations.

The declaration `DMAStream`<*elem_size*> *stream_name* declares a DMA stream with a fresh DMA tag, with the capacity to store *elem_num* elements of size *elem_size* (where the *elem_num* parameter may vary at run-time). A fresh tag means that simultaneous DMA requests for multiple streams can be issued simultaneously and managed independently. The *stream_name*`.start(`*base_address*`)` operation issues and waits for a DMA operation to copy *elem_size* × *elem_num* bytes of data into the *current* buffer for the stream. A DMA request to fill the other buffer with the next *elem_size* × *elem_num* bytes of data is also issued, but not waited for. The *stream_name*`.read(`*offset*`)` first checks that the current buffer contains sufficient data to return the element at the specified offset. If not, this data will reside in the other buffer, so the pending DMA request to fill the other buffer is waited for. A new DMA request is issued to re-fill the *current* buffer, and the buffer pointer is switched. Now that the required data is ready, *elem_size* bytes are copied from the *current* buffer and returned. To avoid stack corruption, the compiler must ensure that all pending DMA operations are completed before returning from a sieve function which uses DMA streams.

[2] For readability, we use C++ template notation to describe streams. Note that streams are generated in the Sieve compiler intermediate representation, and are not exposed to the programmer as our example suggests.

3.2 Pre-fetching Global Variables

Consider the global variable `Size` used to control the **`for`** loops in Listing 1. Since `Size` is in main memory, its value must be fetched using a software cache read. Although this is more efficient than simply fetching `Size` by DMA, `Size` is accessed many times which results in frequent cache reads. The sieve semantics allow us to optimise further by *hoisting* the read from `Size` to the start of the sieve block. This is safe even though `Size` could be modified by the call to `rNormalised`: because `rNormalised` is a sieve function, any potential write to `Size` would be delayed.

Listing 2 illustrates this optimisation: the first executable statement in the sieve block copies the value of `Size` to a local variable `LocalSize`, and all reads from `Size` in the sieve block are changed to read from `LocalSize`.

3.3 Combining Delayed Writes

A write to global memory occurring in a sieve scope causes a call to a `delayedStore` function in the SPU run-time system. As discussed in §2.2, this function appends a side-effect node of the form (*PPU address*, *size*, *data*) to a local queue, streaming the local queue to PPE memory when full. A sieve scope with many side-effects will cause the SPE side-effect queues to fill up quickly, resulting in frequent DMA transfers to stream side-effects to main memory. In addition, at the end of the sieve block, the PPE must commit this large number of side-effects to memory individually.

If these side-effects are the result of small delayed writes (*e.g.* 4 bytes or less) then the 8 byte *PPE address* + *size* overhead associated with each side-effect may be the main reason why the local side-effect queue fills up. In many practical examples, delayed writes are to contiguous memory locations, *e.g.* elements of an array in global memory. Consider the delayed store to `Forces[i]` in the molecular dynamics code of Listing 1. It is clear that each iteration of the outer loop will write to an element of `Forces`. Since the loop uses stride 1 access, these writes are contiguous.

The Sieve-Cell system uses a run-time optimisation to take advantage of commonly occurring contiguous delayed writes. Suppose the `delayedStore` function is called with the side-effect node $(addr_n, size_n, data_n)$, and that the last side-effect in the queue is $(addr_{n-1}, size_{n-1}, data_{n-1})$. The run-time checks whether $addr_n = addr_{n-1} + size_{n-1}$. If this is the case then the new side-effect can be handled by replacing the last side-effect node with $(addr_{n-1}, size_{n-1} + size_n, data_{n-1}{+}{+}data_n)$, where ++ denotes concatenation of bits. Otherwise, the new side-effect node is added to the queue as usual. The contiguity check bears a (small) run-time overhead, and while effective when delayed stores *are* contiguous, may be onerous when delayed stores are fragmented.

We could extend the check to eliminate redundant side-effects by spotting cases where multiple stores are made to the same memory location; we have not found this to be a useful optimisation for practical examples.

4 Experimental Evaluation

4.1 Experimental Setup

We present experimental data for single-precision benchmark programs implemented in standard C++, Sieve C++, and OpenMP C++. The Sieve versions of the programs were

developed from serial base codes by the addition of Sieve annotations, and elimination of global variable updates within kernels. In all cases, a small number of preprocessor macros were then sufficient to allow OpenMP, Sieve, and serial code to coexist in one set of files. We also present results for standard and Sieve C++ versions of three further programs which, due to limitations of the (Alpha version of) XL C++, we were not able to implement using OpenMP; these benchmarks are marked † in the following list:

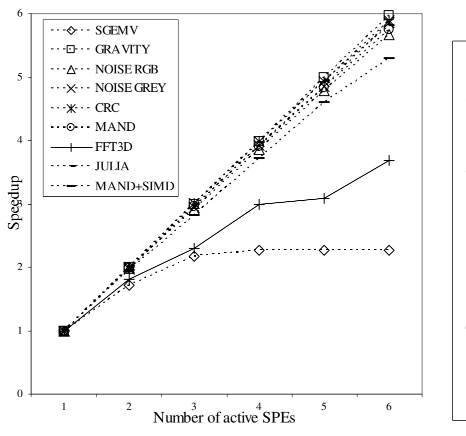
SGEMV: Matrix-vector multiply (8192×4096)
GRAVITY: An N-body molecular dynamics simulation of 8192 particles
NOISE RGB: Noise reduction filter applied to a 512×512 colour image
NOISE GREY: Noise reduction filter applied to a 512×512 greyscale image
CRC: Cyclic redundancy check on a random 8M (1M=2^{20}) word message
MAND: Calculates a 1024×1024 fragment of the Mandelbrot set
FFT3D: Fast Fourier transform of a complex 128^3 data set†
JULIA: Ray traces a 512×512 3D slice of a 4D quaternion Julia set†
MAND+SIMD: The MAND program optimised using SPU SIMD intrinsics†

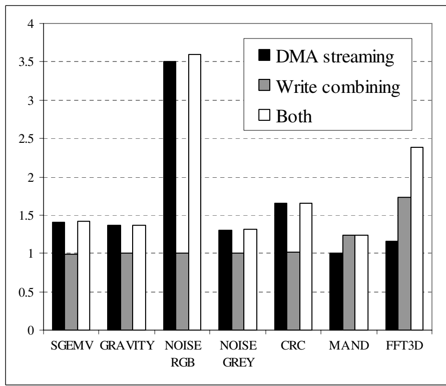
Experiments are performed on a Sony PlayStation 3 console (on which only six of the SPEs are available to the programmer), running Fedora Core 7 Linux, with IBM SDK v3.0.0. The OutputC code produced by the Sieve compiler is compiled using `ppu-gcc` and `spu-gcc`, and Sieve results are compared against programs in standard C++ compiled with `ppu-g++` (all GNU compilers are v4.1.1). OpenMP examples are compiled using the IBM XL C/C++ Single-Source compiler (Alpha Edition, v0.9), and compared against serial code compiled with the same compiler, using the `-qnosmp` to specify that OpenMP directives should be ignored. Both serial versions run on the PPU.

Figure 1(a) plots the speedup of each Sieve C++ program relative to a single SPU when data movement optimisations are applied. Figure 1(b) demonstrates the effectiveness of these optimisations for code parallelised over 6 SPEs, showing speedups for each optimisation and for their combination relative to performance without data movement optimisations. Figure 2 shows the speedup of each Sieve/OpenMP application relative to serial code compiled with `ppu-g++`/XL C++. For the Sieve applications, the number of SPUs is indicated above the corresponding bar. For OpenMP, the number indicated is the number of active parallel threads, which ranges from 1 to 7 since the IBM OpenMP implementation supports parallelisation across the SPUs *and* PPU. The rules as to when a thread is spawned on the PPU rather than an SPU are undocumented; the results indicate that the distribution strategy varies between input programs.

4.2 Discussion

Figure 1(a) shows that six of our nine benchmarks scale almost linearly as the number of active SPUs increases, with GRAVITY showing the best scaling. Of the remaining three benchmarks, MAND+SIMD scales slightly worse than MAND: the gain from using vector intrinsics is high for a small number of SPUs, but more active SPUs leads to higher bus traffic due to side-effect transfer. This slows down the SPUs, making the impact of vector instructions less significant. Analysis using an in-house Codeplay profiler attributes the reasonable but sub-linear FFT3D scaling and the poor SGEMV





(a) Scaling w.r.t. one SPE (b) Effectiveness of Sieve-Cell optimisations

Fig. 1. Benchmark results for Sieve C++ programs

scaling to high bus activity. These benchmarks involve a high rate of data movement in small chunks both to and from SPU local store.

Our data movement optimisations provide no performance improvement for JULIA and MAND+SIMD, which are therefore not shown in Figure 1(b). DMA streaming does not apply to these benchmarks (or to MAND) since they do not read data from global memory. Delayed write combining is effective for MAND and FFT3D, which involve frequent delayed stores. MAND+SIMD involves fewer delayed stores than MAND since the former benchmark writes back pixels in vector chunks. For NOISE RGB, delayed write combining has little effect alone but provides an improvement when combined with streaming. This works the other way for FFT3D, where streaming provides a more significant speedup when applied with write combining.

The results of Figure 2 show that automatic parallelisation using Sieve and OpenMP can lead to significant speedups over PPU-only code, and thus commends both systems. The Sieve results are competitive with those for OpenMP, showing better scaling for three benchmarks. Both approaches perform poorly compared with serial SGEMV. For the GRAVITY benchmark, although scaling is good, the performance of Sieve code with six SPUs is roughly equal the performance of serial code; OpenMP code using all seven parallel threads runs 1.3 times faster than the XL C++ serial version. Of the Sieve-only benchmark programs, MAND+SIMD shows excellent scalability, approaching the performance of hand-written parallel Cell code. Nevertheless, our 4-pixel SIMD execution path prevents an ideal quadrupling of performance. A modest 1.6 times speedup with 6 SPUs is achieved for JULIA. As with GRAVITY, Sieve code for FFT3D with 6 SPUs runs at roughly the same speed as the serial `ppu-g++` code. Further investigation of the weaker speedup responses has shown that programs with a high rate of data transfer between SPE and PPE memory are most strongly affected, due to the cost of DMA operations and the attendant overhead of the delayed store function. The final writeback of side-effects by the PPE does *not* make a significant contribution.

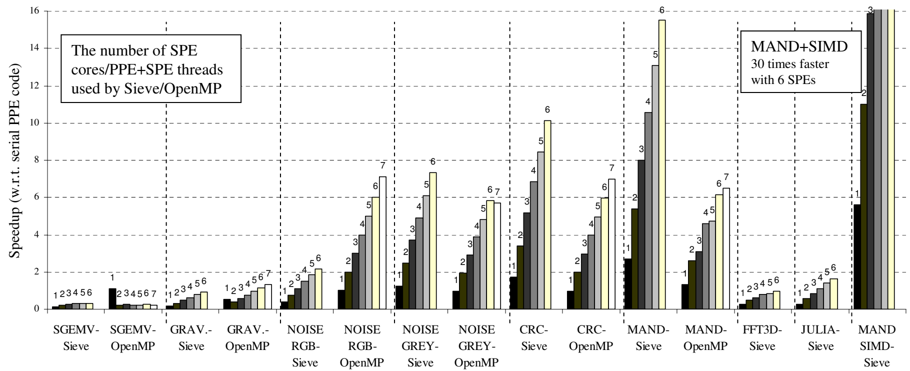


Fig. 2. Scaling for Sieve and OpenMP benchmarks with respect to serial PPE versions

5 Conclusions and Future Work

We have presented the Sieve-Cell system – an auto-parallelisation system for the Cell BE processor based on the Codeplay Sieve C++ language. We have described compile-time and run-time data movement optimisations, and presented experimental results which show the effectiveness of these optimisations on a number of benchmarks, as well as the scaling of parallel code over multiple SPUs. Our experimental results also show that Sieve is competitive with IBM's OpenMP implementation for Cell.

Future work includes comparing the Sieve-Cell system against Sequoia and CellSs on a number of representative benchmarks, and performance comparisons with other architectures. Similar experiments on double-precision data using PowerXCell8i will also be attempted. The design and implemention of advanced data movement techniques for complex access patterns is also underway, as is the development of an overlay system to allow larger applications to be parallelised over SPUs.

Acknowledgements. Thanks to all at Codeplay for their work on the Sieve-Cell system, Mike Houston for providing Sequoia benchmark source code, and the anonymous reviewers for their comments which have helped to improve the paper.

References

1. Bellens, P., Perez, J., Badia, R., Labarta, J.: CellSs: a programming model for the Cell BE architecture. In: Supercomputing 2006, p. 86. ACM Press, New York (2006)
2. Donaldson, A.F., Riley, C., Lokhmotov, A., Cook, A.: Auto-parallelisation of sieve C++ programs. In: Bougé, L., Forsell, M., Träff, J.L., Streit, A., Ziegler, W., Alexander, M., Childs, S. (eds.) Euro-Par Workshops 2007. LNCS, vol. 4854, pp. 18–27. Springer, Heidelberg (2008)
3. Fatahalian, K., et al.: Sequoia: programming the memory hierarchy. In: Supercomputing 2006, p. 83. ACM Press, New York (2006)

4. Hofstee, H.P.: Power efficient processor architecture and the Cell processor. In: HPCA 2005, pp. 258–262. IEEE Computer Society, Los Alamitos (2005)
5. Lokhmotov, A., Mycroft, A., Richards, A.: Delayed side-effects ease multi-core programming. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 641–650. Springer, Heidelberg (2007)
6. Valiant, L.: A bridging model for parallel computation. Commun. ACM 33(8), 103–111 (1990)

A Unified Runtime System for Heterogeneous Multi-core Architectures

Cédric Augonnet and Raymond Namyst

INRIA Bordeaux – LaBRI – University of Bordeaux
cedric.augonnet@inria.fr, raymond.namyst@labri.fr

Abstract. Approaching the theoretical performance of heterogeneous multicore architectures, equipped with specialized accelerators, is a challenging issue. Unlike regular CPUs that can transparently access the whole global memory address range, accelerators usually embed local memory on which they perform all their computations using a specific instruction set. While many research efforts have been devoted to offloading parts of a program over such coprocessors, the real challenge is to find a programming model providing a unified view of all available computing units.

In this paper, we present an original runtime system providing a high-level, unified execution model allowing seamless execution of tasks over the underlying heterogeneous hardware. The runtime is based on a hierarchical memory management facility and on a *codelet* scheduler. We demonstrate the efficiency of our solution with a LU decomposition for both homogeneous (3.8 speedup on 4 cores) and heterogeneous machines (95 % efficiency). We also show that a "granularity aware" scheduling can improve execution time by 35 %.

1 Introduction

The last years have witnessed the tremendous invasion of multicore architectures in the field of parallel computing. Although many research efforts have been devoted to designing multicore-aware algorithms and software, application developers are still having a hard time trying to get the most of these hierarchical architectures. Unfortunately, the situation is about to get even worse, with the emergence of a new architecture trend: heterogeneous multicore architectures.

We have known for a long time that making use of specific hardware accelerators can dramatically speed up applications featuring data-parallelism. The novelty is that such hardware is now available off the shelf: clusters featuring GPGPUs, FPGAs or even Cell processors are affordable for most users. It is likely that many supercomputers will be equipped with such heterogeneous hardware in the future. Consequently, programmers now have to deal with architectures composed by a mix of regular CPUs and specific PUs (SPUs, GPGPUs).

Only few applications are currently taking advantage of these hybrid architectures. And even fewer applications are able to use both types of computing units at the same time. Many approaches actually only rely on offloading

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 174–183, 2009.
© Springer-Verlag Berlin Heidelberg 2009

parts of the computations over accelerators, since there exists no unified runtime system allowing programmers to seamlessly exploit all available computing units. Thus, the most widely used programming models on such machines are a mix of pthreads and CUDA [3] (for nVidia GPGPUs) or libspe (for Cell's SPUs). Of course, higher level approaches exist (RapidMind [14], CellSs [9], HMPP [8]) that hide the hardware complexity to programmers. However, they currently only support a single type of accelerator at a time. To deal with irregular application, where dynamic load balancing is required, all these systems rely on specific runtime systems that implement memory transfers between main memory and the accelerators.

In this paper, we present the design of a unified runtime system that provides a simple yet powerful interface to exploit multiprocessors equipped with heterogeneous accelerators (e.g. SPU+GPU+CPU). The programming model is based on a high level memory management interface enabling hierarchical description of data domains. Applications tasks are running as "*codelets*" over the underlying hardware computing units: the runtime system automatically takes care of gathering input data before their execution and scattering output results upon completion. Thanks to the concept of divisible tasks, our approach helps to develop programs that dynamically adapt to the available processing units.

2 A Uniform Approach to Exploiting Heterogeneous Computing Units

Designing a runtime system for *heterogeneous* multicore machines, featuring different accelerators and computing units, introduces challenging issues. Traditional shared-memory homogeneous multicore architectures are either programmed using high-level languages (*e.g.* OpenMP [1], Cilk [11]) or low-level task management libraries (*e.g.* pthread, libspe). In both approaches, the underlying runtime system is merely devoted to task scheduling. In contrast, heterogeneous machines have much more runtime requirements since they do not provide a coherent (nor even shared) global memory. Unlike regular CPUs that can transparently access the whole global memory address range, accelerators often embed local memory on which they perform all their computations. With no help from the runtime system, programmers would have to explicitly enforce memory consistency between accelerators, which would seriously impact both programmability and portability as we potentially have to deal with various kinds of accelerators (e.g. Cell's SPU with GPGPUs). This definitely emphasizes the need for a runtime featuring high-level memory management interface.

Besides, using the *thread* concept (as provided by modern operating systems) does not allow to efficiently exploit those architectures. Indeed, tasks running over specialized computing units such as GPGPUs can only use a specific instruction set and access a private memory address space. The "*codelet*" concept, which models non-preemptible offloadable tasks, has been recognized to be a much more appropriate execution model [8]. However, provided the variety of accelerator technologies and the gap between their respective performance,

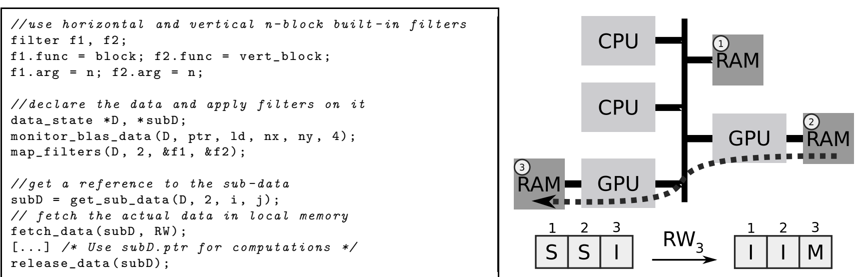
statically determining a relevant granularity is a major issue. Therefore, we believe that the runtime system should offer support for a dynamically adaptive grain size, in collaboration with the programmer who would be responsible for submitting splittable tasks.

2.1 A Topology Aware Hierarchical Data Management

The major component of our runtime system is a data management facility, which aims at providing a high level API that hides the complexity raised by accelerators' heterogeneity. The idea is to abstract the description of memory regions in such a way that the underlying data transfers and data caching policies can be optimized, depending on the hardware capabilities.

Hardware accelerators typically work only on a subset of data at a time. As it would be unwise to manipulate a huge matrix while only working on a subset of its elements, our library features an interface to manipulate sub-data as well. However, maintaining an MSI protocol on several sub-data is complex as they could overlap. In that case, modifying a piece of data requires to invalidate all copies of the overlapping chunks. This requires to maintain an intersection graph, and makes the coherence protocol NP-complete. Constructing such a graph without an expressive description of data subsets is also costly if not impossible. To address those issues, we introduce the notion of *filter* as a result of two simple observations. First, the programmer is usually well aware of the data layout which the library should therefore not have to infer. Second, we can reasonably make sure all sub-data are disjointed, breaking ties by making new sub-data out of intersections. Contrary to HPF, filters are not restricted to dense matrix structures and can be used with multiple data interfaces (*e.g.* CSR).

Our runtime features a number of predefined filters (e.g. block-cyclic distribution), and user-provided filters can easily be added if needed. All filters can be applied recursively, so that data is partitioned following a tree-based hierarchy. Applications must only access the leaves of the resulting tree to maintain consistency of each sub-data independently. This restriction is consistent with a



```
//use horizontal and vertical n-block built-in filters
filter f1, f2;
f1.func = block; f2.func = vert_block;
f1.arg = n; f2.arg = n;

//declare the data and apply filters on it
data_state *D, *subD;
monitor_blas_data(D, ptr, ld, nx, ny, 4);
map_filters(D, 2, &f1, &f2);

//get a reference to the sub-data
subD = get_sub_data(D, 2, i, j);
// fetch the actual data in local memory
fetch_data(subD, RW);
[...] /* Use subD.ptr for computations */
release_data(subD);
```


Fig. 1. Internal API to manipulate a sub-data

Fig. 2. Coherence protocol

data-parallelism paradigm where all computations are performed locally. When the application needs to access an inner node of the tree, for instance during a *reduction* phase, we temporary unapply some filters. Filters can be efficiently restored later thanks to a lazy design that makes this operation virtually costless.

Once appropriate filters are created and applied by the application, the memory management facility is typically invoked each time a task is scheduled somewhere, to make sure that the required input data is made available on time (see Fig. 1). A straightforward "*write-through*" implementation would simply maintain the up-to-date state of each data in main memory, but this would require numerous unnecessary data transfers between main memory and embedded memory banks. In contrast, the use of a *write-back* model would maintain memory consistency in a lazy fashion (i.e. data transfers are only performed when actually needed) and would thus reduce the stress on the memory buses.

From a technical perspective, implementing such a memory model is challenging, as one cannot assume that it is feasible to directly move data between any combination of accelerators. We thus have extended the pure *write-back* model so that we sometimes allow data modifications to be propagated eagerly to main memory. To enforce data consistency, we maintain a list of the nodes that are holding a valid copy of each data. To this end, we use a similar approach to the one used by *write-back* cache coherency protocols implemented within SMP machines. Since our approach does not rely on any specific hardware support, we use a *directory-based* protocol. As shown on Figure 2, every data is associated to a vector indicating its state on each memory node. There are three states, *invalid* (I) that indicates the node does not hold a valid copy, *modified* (M) for nodes that do have one, and *shared* (S) if there are several nodes with the data. Along with the updating of those states, the MSI automaton also describes which data transfers are needed.

When accessing data for the first time, computing units have to allocate a buffer of sufficient size. The runtime system maintains a reference count of units that actually use the data, so that deallocations are performed in a lazy manner. When a unit accesses a data larger than its remaining memory, some of the unused local data is either discarded or flushed to main memory, depending on the existence of another valid copy of the data. Note that any other unit holding enough free memory is eligible for that purpose. This memory reclaiming mechanism thus allows to transparently handle data sets which size would not entirely fit within accelerators memory. As the size of embedded memory typically vary from orders of magnitude (from 256 KB on a SPU to 1 GB on a GPU), this is crucial when running the same application on various architectures.

2.2 Modeling Tasks with Codelets

Given the aforementioned restrictions imposed by some hardware accelerators about memory accesses, our unified execution model relies on the use of *codelets*, which are tasks enriched with a description of their input and output data. This description is performed with the high-level data library we defined in the previous section. While being a constraint for the programmer, the need to

precisely specify which data are used opens room for numerous optimizations. It is for instance possible to take data affinity into account while scheduling. Figure 3 illustrates the *codelet* structure used by our runtime. To execute a *codelet*, each eligible accelerator is given a function specific to its architecture. The corresponding "*kernels*" are written by the programmer, but we expect compiling environments to be able to generate them automatically from a generic source code.

The execution of *codelets* is asynchronous and there is no guarantee about their ordering. We therefore added the possibility to perform a *callback* function on the host after the termination of a *codelet*. If synchronization is needed, these callbacks can perform interactions between accelerators and the host. Such interactions can involve costly operations that may decrease the entire system performance, possibly overwhelming the actual *codelet* computation time. The callback mechanism is therefore not sufficient to enforce task dependencies. Hence, our runtime features facilities to express dependencies between tasks within the *codelet* structure itself. This makes programming easier and gives more freedom to the scheduler. It is indeed simpler to extract parallelism when the runtime has a wider view of the task and data dependencies.

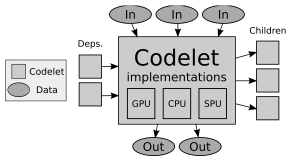


Fig. 3. The codelet structure

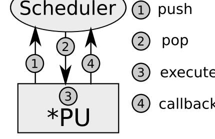


Fig. 4. Execution model

Adding a driver to support a new accelerator architecture only requires to write a limited number of functions as shown on Figure 4. First, the driver needs to supply methods to *fetch* and to *push codelets* from the scheduler which is a mere list of *codelets* from the driver's perspective. This is straightforward on a CPU or a GPU that is controlled directly from the host, but it requires a little more work on a CELL's SPU which has to use DMA mechanisms to manipulate the *codelet* list. The driver must also supply a method that executes the code associated to its architecture in the *codelet* structure. This may involve to actually upload the executable code into the memory of the accelerator, possibly using our data library. Last, the driver needs to offer a mechanism to schedule the execution of a *codelet*'s callback on the host after its termination. On the CELL, we use special hardware interrupts from the SPUs to handle a *codelet* termination and execute its callback.

2.3 Discussion

With respect to our portability requirements, supporting a new architecture is relatively simple. In addition to the limited number of methods needed to

execute and manipulate *codelets* on an accelerator, we have to write functions that perform the actual memory transfers with main memory. As memory bandwidth is a scarce resource, we otherwise believe that it is worth writing some optional methods to transfers data directly between some pairs of accelerators, even though there is a slight risk for a combinatorial explosion as the number of supported architectures grows. Provided that our only assumption is that an accelerator needs a mechanism to access main memory, we are confident that our model is well suited for most current and future heterogeneous architectures.

We also believe that our execution model should be helpful for various purposes. On the one hand, the expressiveness of our high-level interface helps programming experts to transmit useful indications to the scheduler while getting some feedback in return. On the other hand, although our solution does not address the code generation problem, compilers can infer most task dependencies, and generate *codelets* that our runtime can schedule efficiently.

3 Experiments

We perform our experiments on a E5410 Xeon quadcore running at 2.33 GHz with 4 GB of memory. This machine also has a nVidia Quadro FX4600 CUDA-enabled graphic card with 768 MB of embedded memory.

3.1 Programmability

Our first contribution is to ease the programming of heterogeneous architectures by the mean of high-level abstractions. This is illustrated on the common matrix multiplication example which follows. After partitioning data into blocks as shown on Figure 1, we actually launch *codelets* as shown below.

```
for (i = 0; i < nslicesx; i++) {
    for (j = 0; j < nslicesy; j++) {
        codelet *cl = malloc(sizeof(codelet));
        cl->where = ANY;     cl->core_func = core_mult;
                               cl->cublas_func = cublas_mult;
        cl->cb = callback_func; cl->argcb = &jobcounter;
        cl->nbuffers = 3; [...]
        cl->buffers[2].state = subdata_ref(&C, 2, i, j);
        cl->buffers[2].mode = W; /* write results into submatrix C_ij */
        push_task(cl); /* schedule the codelet */
    }
}
```

Writing the actual blocked parallel matrix multiplication is then immediate as each *codelet* computes one block of the output matrix. A codelet takes A_i and B_j as inputs and writes its result into $C_{i,j}$, which is described naturally with our high-level library. As there are no task dependencies, the callback only decrements a task counter until there is no more work. Those *codelets* can either be executed on a GPU by calling `cublas_mult` or on a CPU with `core_mult`.

```
void cublas_mult(buffer_descr *descr, void *arg) {
    cublasSgemm(..., descr[2].nx, descr[2].ny ...);
}

void core_mult(buffer_descr *descr, void *arg) {
    cblas_sgemm(..., descr[2].ny, descr[2].nx, ...);
}
```

The programmer must write the kernels that run on the various resources. But the `descr` array is automatically filled with a description of all buffers : the first matrix is for instance located at address `descr[0].ptr`. All the underlying data transfers are transparent, making it much easier to concentrate on the algorithm itself. It is also worth noting that the programmer does not need to take into account the possibility of solving a problem larger than GPU memory.

3.2 Heterogeneous Computing

We now validate our core affirmation that it is possible to efficiently use multiple heterogeneous resources by comparing the performance depending on the available computational resources. This experiment consists in multiplying two 16384×16384 single precision matrices. We compute a synthetic performance metric out of the measured execution times.

Table 1 not only shows our system performs well with multiple homogeneous cores as we get a 3.8 speedup on four cores, but we also transparently exploit an hybrid architecture, only supplying the implementation of the kernel on various resources. It appears that we need to devote a core to control the accelerators. On the one hand, the host must remain reactive to external events; on the other hand, such events are especially harmful for cache-sensitive computations such as BLAS. Our runtime system therefore obtains 82.47 GFlops with three cores and a GPU, which is 95 % of the added performance of either three cores (25.24 GFlops) and a single GPU (62.06 GFlops).

Table 1. Combining heterogeneous resources

	1 core	3 cores	4 cores	4 cores / 1 GPU	3 cores / 1 GPU	1 GPU
GFlops	8.70	25.24	32.83	62.34	82.47	62.06

3.3 Extracting Enough Parallelism

We implemented the LU decomposition presented on the Algorithm 1 twice to underline the importance of a collaboration between the programmer and the runtime system in order to improve scheduling. More precisely, those two versions share the same implementation of the kernels, but the callbacks differ in their handling of the dependencies between *codelets*. In the first version, the *codelets* of type D are not scheduled until all B and C *codelets* are finished, and the A *codelet* waits for all D ones. This means that the entire algorithm suffers from a sequential section, which is critical for performance. In the second

Algorithm 1. Blocked LU decomposition of matrix M

for $k = 1$ *to* n **do**
  Decompose $M_{k,k}$; `/* 1 Codelet A */`
  for $i = k + 1$ *to* n **do**
    Find $M_{i,k}$ with $M_{k,k}M_{i,k} = M_{i,k}$; `/*` $(n - k - 1)$ `Codelets B */`
  for $j = k + 1$ *to* n **do**
    Find $M_{k,j}$ with $M_{k,j}M_{k,k} = M_{k,j}$; `/*` $(n - k - 1)$ `Codelets C */`
  for $i = k + 1$ *to* n **do**
    for $j = k + 1$ *to* n **do**
      $M_{i,j} -= M_{i,k}M_{k,j}$; `/*` $(n - k - 1)^2$ `Codelets D */`

implementation, *codelets* are scheduled as soon as data dependencies are verified. This has a clear effect on the amount of parallelism of the overall algorithm as shown on Figure 5 : while most resources are almost stalled during the execution of the first implementation, they keep running nearly all the time in the second version, substantially reducing the execution time by 15 %.

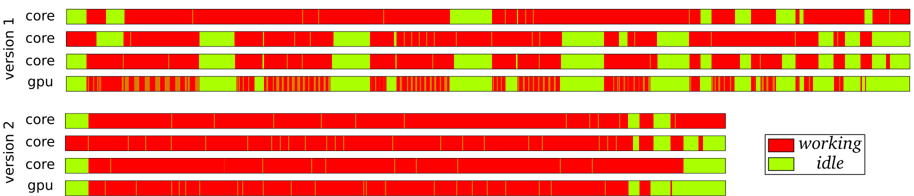


Fig. 5. Supplying enough parallelism helps to reduce load imbalance

Table 2. Optimizing the LU decomposition

Optimisation	Reference	Memory pinning	Dependencies	Priorities
GFlops	49.65	53.62	64.94	67.33
Gain (%)	0	8.0	30.9	35.6

There remains load balancing issues at the end of the execution which result from algorithm's inherent lack of parallelism. To avoid that, we added the notion of priority tasks to schedule *codelets* of type A – and all their direct dependencies – as soon as possible. In addition to the use of registered memory that reduces the need for costly memory copies within the CUDA driver, we see another 5 % improvement using priorities on Table 2.

4 Related Work

Programmable GPUs can be controlled using specific languages (*e.g.*, Cg, HLSL or GLSL). Others require less knowledge of graphic APIs using higher-level

abstractions like *streams* instead of graphical primitives (*e.g.* Brooks, Scout, Glift). Given that successful evolution toward GPGPUs, well summarized by Ownens *et al.* [17], constructors enriched hardware with generic features and now provide generic programming environments such as CUDA [3], CAL [2]. As summarized by Buttari *et al.* [6], Cell programming usually consists in the use of low-level specific mechanisms provided by the LibSPE, even though there are higher level abstractions either in runtime systems (*e.g.* ALF [7], MCF [4], Charm++ [12], Gordon [15]) or at compile time (*e.g.* Octopiler).

There are also substantial efforts to develop hybrid programming models which differ in the objects they manipulate. On the one hand, there are *data parallel* approaches which map operations on arrays or matrices (*e.g.* RapidMind [14], Brooks [5], PeakStream). On the other hand, some follow a *task parallelism* model, and offer architecture independent abstractions for offloadable functions (*e.g.* Merge [13], Sequoia [10]). A growing number of compiler frameworks are also intended to offer support for heterogeneous architectures (*e.g.* HMPP [8], EXOCHI [13], R-Stream [20]). Likewise, high level asynchronous stream processing systems such as AETHER's S-Net [19] or the Scalp project [15] rely on support at the runtime level. Given the lack of an actual hybrid programming standard, substantial efforts are done to adapt main standards like MPI [18,16], or OpenMP that CellSs extends to express task and data dependencies [9], which is interesting to generates *codelets*.

5 Conclusion and Future Work

In this paper, we present a runtime system that transparently handles the coherency of hierarchical data structures over heterogeneous multiprocessor machines. With the support of the *codelet* abstraction to model tasks, we sketch the basis of a generic task scheduling platform. We show that our prototype actually obtains very good performance for non-trivial problems, either on accelerators or on hybrid architectures. Those abstractions indeed offer the opportunity to drive accelerator programming beyond the mere solving of technical issues, hence allowing to concentrate on the algorithmic issues. Another conclusion is the importance of a proper task scheduling, and the need for an expressive interface.

In addition to a seamless use of multiple GPU, we plan to make our data library asynchronous which should be profitable in the context of scheduling policies using data prefetching. Besides improving the on-going support of the Cell, we will investigate *lock-free* protocols to prevent scalability concerns. On a longer term, we envision to supply a set of scheduling policies covering a wide spectrum of problematics, going from a better support of NUMA machines to an inter-node scheduling using MPI. We claim that our runtime could provide the necessary support that compilation environments and specialized libraries lack to harness the growing complexity of heterogeneous machines. With performance portability as a major goal, and given the urgent need for a standardization of all the work around heterogeneous multicore programming, our runtime could contribute to the efforts made around OpenMP and other high level approaches.

References

1. http://www.openmp.org/
2. AMD FireStream SDK, http://ati.amd.com/technology/streamcomputing/
3. Cuda zone, http://www.nvidia.com/cuda
4. Bouzas, B., Cooper, R., Greene, J., Pepe, M., Prelle, M.J.: Multicore framework: An api for programming heterogeneous multicore processors. In: STMCS. Mercury Computer Systems (2006)
5. Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., Hanrahan, P.: Brook for gpus: stream computing on graphics hardware. In: SIGGRAPH 2004 (2004)
6. Buttari, A., Luszczek, P., Kurzak, J., Dongarra, J., Bosilca, G.: A rough guide to scientific computing on the playstation 3. Technical report, UTK (2007)
7. Crawford, C.H., Henning, P., Kistler, M., Wright, C.: Accelerating computing with the cell broadband engine processor. In: CF 2008 (2008)
8. Dolbeau, R., Bihan, S., Bodin, F.: HMPP: A hybrid multi-core parallel programming environment (2007)
9. Duran, A., Perez, J.M., Ayguade, E., Badia, R., Labarta, J.: Extending the openmp tasking model to allow dependant tasks. In: Eigenmann, R., de Supinski, B.R. (eds.) IWOMP 2008. LNCS, vol. 5004, pp. 111–122. Springer, Heidelberg (2008)
10. Fatahalian, K., Knight, T.J., Houston, M., Erez, M., Reiter Horn, D., Leem, L., Young Park, J., Ren, M., Aiken, A., Dally, W.J., Hanrahan, P.: Sequoia: Programming the memory hierarchy. In: Supercomputing (2006)
11. Frigo, M., Leiserson, C.E., Randall, K.H.: The implementation of the cilk-5 multithreaded language. In: PLDI 1998 (1998)
12. Kunzman, D., Zheng, G., Bohm, E., Kalé, L.V.: Charm++, Offload API, and the Cell Processor. In: PMUP 2006 (2006)
13. Linderman, M.D., Collins, J.D., Wang, H., Meng, T.H.: Merge: a programming model for heterogeneous multi-core systems. In: ASPLOS XIII (2008)
14. McCool, M.D.: Data-parallel programming on the cell be and the gpu using the rapidmind development platform (2006)
15. Nijhuis, M.: Simple and Efficient Parallel Streaming Applications (working title). Ph.D thesis, Vrije Universiteit Amsterdam (2008) (to appear)
16. Ohara, M., Inoue, H., Sohda, Y., Komatsu, H., Nakatani, T.: Mpi microtask for programming the cell broadband enginetm processor. IBM Syst. J. 45(1) (2006)
17. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26(1), 80–113 (2007)
18. Pakin, S.: Receiver-initiated Message Passing over RDMA Networks. In: IPDPS 2008 (2008)
19. Penczek, F.: Design and Implementation of a Multithreaded Runtime System for the Stream Processing Language S-Net. Master's thesis, Institute of Software Technology and Programming Languages, University of Lübeck, Germany (2007)
20. Meister, B., Lethin, R., Leung, A., Schweitz, E.: R-stream: A parametric high level compiler. In: HPEC (2006)

(When) Will CMPs Hit the Power Wall?

Cor Meenderinck and Ben Juurlink

Computer Engineering Department,
Delft University of Technology
Delft, the Netherlands
{cor,benj}@ce.et.tudelft.nl

Abstract. The power wall is currently one of the major obstacles computer architecture is facing. In this paper we analyze the impact of the power wall on CMP design. As a case study we model a CMP consisting of Alpha 21264 cores, scaled to future technology nodes according to the ITRS roadmap. When running at the maximum clock frequency, such a CMP would far exceed the power budget. Although power limits performance significantly, technology improvements will still provide performance growth. Amdahl's Law highly threatens this performance growth, but might not be valid for all application domains. In those cases Gustafson's Law could be valid which is much more optimistic. From our results we derive some principles to prevent CMPs from hitting the power wall.

1 Introduction

It is commonly believed that we have reached the *power wall*, meaning that uniprocessor performance improvements have come to an end due to power constraints. The main causes of the increased power consumption are higher clock frequencies and power inefficient techniques to exploit more Instruction Level Parallelism (ILP), such as wide-issue superscalar execution. Hitting the power wall is also one of the reasons why industry has shifted towards multicores or chip multiprocessors (CMPs). Because CMPs exploit explicit Thread Level Parallelism (TLP), their cores can be simpler and do not need additional hardware to extract ILP. In other words, CMPs allow exploiting parallelism in a power efficient way.

Figure 1, taken from [1], illustrates the power wall. Uniprocessors have basically reached the power wall. As argued above, multicores can postpone hitting the power wall but they are also expected to hit the power wall. Several questions arise like when will CMPs hit the power wall, what limitation will cause this to happen, what can computer architects do to avoid the problem, etc. It is generally believed that the power efficiency of CMPs can be improved by designing asymmetric or heterogeneous multicores. For example, several domain specific accelerators could be employed which are turned on and off according to the actual workload. But, is the power saving it provides worth the area cost? In this paper we try to answer those questions.

Specifically, in this paper we focus on technology improvements as they have been one of the main drivers of performance growth in the past. According to the

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 184–193, 2009.
© Springer-Verlag Berlin Heidelberg 2009

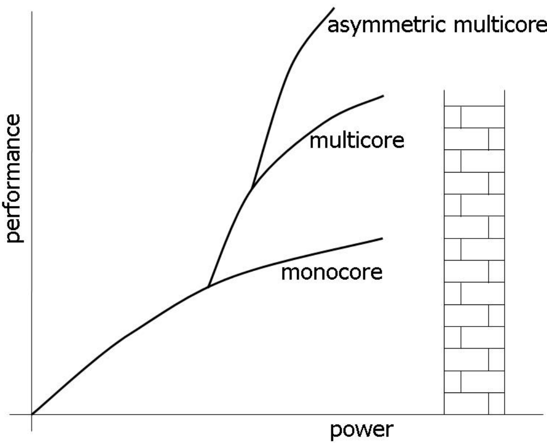


Fig. 1. The power wall problem

ITRS roadmap [2], technology improvements are expected to continue. However, power constraints might not allow us to exploit them fully. In the next section we analyze the limits of performance growth due to technology improvements with respect to power constraints. In Section 3 based on the results of our experiment we conclude that CMPs can offer significant performance improvements provided a number of principles are followed.

Of course our model is necessarily rudimentary. For example, it does not consider modifications in memory and interconnect, bandwidth constraints, nor static power dissipation due to leakage. Nevertheless, the power wall has been predicted, multicores are expected to be the remedy, asymmetric multicores have been envisioned, but to the best of our knowledge this has never been quantified. From our results we hope to derive some principles which can be the basis for future work.

2 Performance and Power of Future Multicores

To analyze the effect of technology improvements on the performance of future CMPs and to investigate the power consumption trend, the following experiment was performed. We take an Alpha 21264 chip (core and caches), scale it to future technology nodes according to the ITRS roadmap, create a hypothetical CMP consisting of the scaled cores, and derive the power numbers. Specifically, we calculate the power consumption of a CMP for full blown operation, i.e., all cores are active and run at the maximum possible frequency. Furthermore, we analyze the performance growth over time if the power consumption is restricted to the power budget allowed by packaging.

The Alpha 21264 [3] core was chosen as subject of this experiment for two reasons. First, the Alpha has been well documented in literature, providing the data required for the experiment. Second, the 21264 is a moderately sized core

lacking the aggressive ILP techniques of current high performance cores. Thus, it is a good representative of what is generally expected to be the processing element in future many-core CMPs. The relevant parameters of the 21264 core for this analysis are the following: year $t_0 = 1998$, technology node $L_{orig} = 350$ nm, supply voltage $V_{orig} = 2.2\ V$, die area $A_{orig} = 314\ mm^2$, dynamic power (@400 MHz) = 48 W, and dynamic power (@600 MHz) = 70 W.

2.1 Scaling of the Alpha 21264

The 21264 is scaled according to data in the 2007 edition of the International Technology Roadmap for Semiconductors (ITRS) [2]. The relevant parameters are given in Table 1. The values of the technology node and the on-chip frequency were taken from Page 79 of the Executive Summary Chapter. The on-chip frequency is based on the fundamental transistor delay, and an assumed maximum number of 12 inverter delays. The die area values were taken from the table on Page 81 of the Executive Summary. Finally, the values of the supply voltage and the gate capacitance (per micron device) were taken from the table starting at Page 11 of the Process Integration, Devices, and Structures Chapter of the roadmap.

Table 1. Technology parameters of the ITRS roadmap

	2007	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**
technology (nm)	68	57	50	45	40	36	32	28
frequency (MHz)	4700	5063	5454	5875	6329	6817	7344	7911
die area (mm^2)	310	310	310	310	310	310	310	310
supply voltage (V)	1.1	1	1	1	1	0.9	0.9	0.9
$C_{g,total}$ ($F/\mu m$)	7.10E-16	8.40E-16	8.43E-16	8.08E-16	6.5E-16	6.29E-16	6.28E-16	5.59E-16
	2015	**2016**	**2017**	**2018**	**2019**	**2020**	**2021**	**2022**
technology (nm)	25	22	20	18	16	14	13	11
frequency (MHz)	8522	9180	9889	10652	11475	12361	13351	14343
die area (mm^2)	310	310	310	310	310	310	310	310
supply voltage (V)	0.8	0.8	0.7	0.7	0.7	0.65	0.65	0.65
$C_{g,total}$ ($F/\mu m$)	5.25E-16	5.07E-16	4.81E-16	4.58E-16	4.1E-16	3.91E-16	3.62E-16	3.42E-16

To model the experimental CMP for future technology nodes, we scale all relevant parameters of the 21264 core. The values that are available in the ITRS, we use as such. The others we scale using the available parameters by taking the ratio between the original 21264 parameter values and the predictions of the roadmap. Below we describe in detail for each scaled parameter how this was done. The gate capacitance of the 21264 was not found in literature, thus we extrapolated the values reported in the roadmap and calculated a value of $1.1 \times 10^{-15} F/\mu m$ for 1998.

First, the area of one core was scaled. Let $L(t)$ be the process technology size for year t and let L_{orig} be the process technology size of the original core. The area of one 21264 core in year t will be $A_1(t) = A_{orig} \times \left(\frac{L(t)}{L_{orig}}\right)^2$. Figure 2 depicts the results for the time frame considered. The area of one core decreases more or

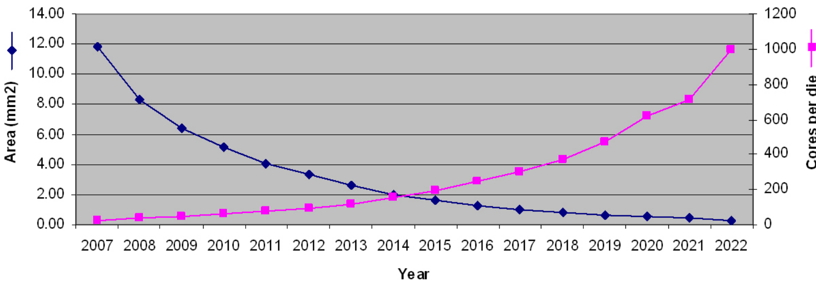


Fig. 2. Area of one core and the number of cores per die

less quadratically over time, and will be about one third of a square millimeter in 2022.

Second, using the scaled area of one core, the number of cores that could fit on a single die was calculated. The ITRS roadmap assumes a die area of $310mm^2$ for the entire time frame. Thus, the total number of cores per die in year t is $N(t) = \frac{310}{A_1(t)} = \frac{310}{A_{orig}} \times \left(\frac{L(t)}{L_{orig}}\right)^2$, which is depicted in Figure 2. The graph shows a doubling of cores roughly every three years which is in line with the expectations [4]. In 2022 our calculations predict a CMP with 999 cores.

Finally, the power of one core was scaled. Power consumption consists of a dynamic and a static part, of which the latter is dominated by leakage. The data required to scale the static power is not available to us and thus we restrict this power analysis to dynamic power. It is expected that leakage remains a problem and thus our estimations are conservative.

The dynamic power is given by $P_{dyn} = \alpha C f V^2$, where α is the transistor activity factor, C is the gate capacitance, f is the clock frequency, and V is the power supply voltage. The activity factor α of the 21264 processor is unknown and also depends on the application, but since this does not change with scaling, it can be assumed to be constant. The capacitance C (F) in the equation is different from capacitance $C_{g,total}$ $(F/\mu m)$ in Table 1, but they relate to each other as $C \propto C_{g,total} \times L$. Thus, the dynamic power at year t is calculated as:

$$P(t) = P_{orig} \times \frac{C_{g,total}(t) \times L(t)}{C_{g,total,orig} \times L_{orig}} \times \frac{f(t)}{f_{orig}} \times \left(\frac{V(t)}{V_{orig}}\right)^2 . \quad (1)$$

This analysis assumes that the cores run at the maximum possible frequency. Figure 3 depicts the power of one core over time. As the curve shows it roughly decreases linearly, resulting in less than 2 W in 2022.

2.2 Power and Performance Assuming Perfect Parallelization

Now that all parameters have been scaled, it is possible to calculate the power consumption of the total CMP. It is assumed that all cores are active and thus $P_{total}(t) = N(t) \times P(t)$. Figure 3 depicts the total power over time and shows that

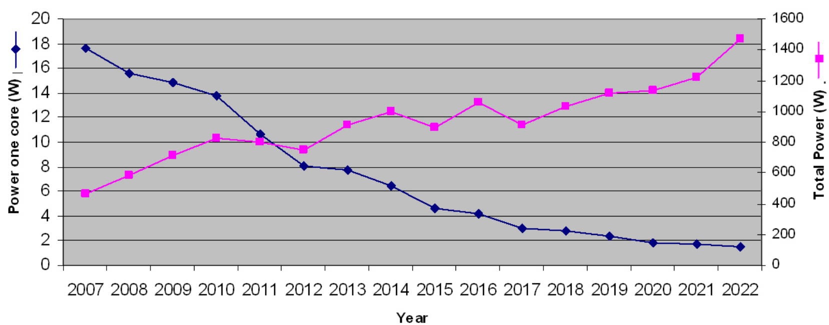


Fig. 3. Power of one core and total power of the case study CMP

the total power consumption of the modelled CMP in 2008 is 600 W, gradually increases, and reaches 1.5 kW in 2022. The roadmap predicts that the power budget allowed due to packaging constraints is 198 W. It is clear that for the entire calculated time span the power consumption of our hypothetical CMP exceeds the power budget. This is why power has become one of the main design constraints nowadays.

The figure also shows that the difference between the power budget and the power consumption of the full blown hypothetical CMP is increasing over time. That means that a large part of the technology improvement cannot be put into effect for performance growth. For example, between 2011 and 2020 technology allows doubling the on-chip frequency. However, the power consumption would increase with a factor 1.5. Thus, for equal power only a small frequency improvement would be possible.

Next we analyze the performance improvement that can be achieved by this CMP. Assuming that the application is perfectly parallelizable, the parameters that influence performance are frequency and the number of cores. In this case the speedup in year t, relative to the original 21264 core, is given by $S(t) = \frac{f(t)}{f_{orig}} \times \frac{N(t)}{1}$. This speedup is depicted in Figure 4 as the non-constrained speedup.

We are interested in the speedup achieved by CMPs that meet the power budget of 198 W. As the results show the power budget is exceeded when all cores are used concurrently at the maximum frequency. Thus, the non-constrained speedup is not achievable in practice. To meet the power budget, either the frequency could be scaled down or a number of cores could be shut down. Since both measures are linear to the speedup, the power-constrained speedup can be defined as:

$$S_{power_constr.}(t) = \frac{f(t)}{f_{orig}} \times \frac{N(t)}{1} \times \frac{P_{budget}}{P_{total}(t)}, \tag{2}$$

where P_{budget} is the power budget allowed by packaging.

Figure 4 depicts the power-constrained performance of the case study CMP over time. To increase readability we normalized the result to 2007. The curve shows a performance growth of 27% per year. Also the non-constrained performance growth is depicted. Note that the latter is growing with 37% per year

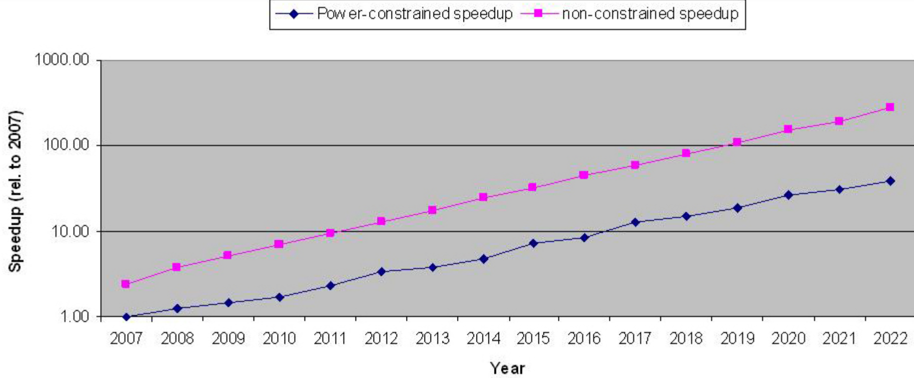


Fig. 4. Power-constrained performance growth

and that the gap between the two is increasing. The robustness of the model to variations in the input parameters is rather good. Even if all parameters we took from the ITRS roadmap vary by 20%, the predicted performance in 2022 varies between 46% and 130%.

To put these results in historical perspective we compare to Figure 2.1 of Hennessy and Patterson [5]. From the mid-1980s to 2002 the graph shows an annual performance growth of 52%. Then from 2002 the annual performance growth dropped to 20%. Considering this historical perspective, the predicted annual performance growth of 27% is a bit on the high side, but not far of. The main conclusion from these results is that although power severely limits performance, substantial performance improvements are still possible using the CMP paradigm.

2.3 Performance Assuming Non-perfect Parallelization

So far we assumed a perfectly parallelizable application. In practice this is not always the case as there might be purely serial code. If this is the case we can apply either Amdahl's Law or Gustafson's Law [6].

Both Amdahl and Gustafson proposed an equation to calculate the speedup achieved by a parallel system. At first sight they look different but Yuan Shi proved that actually they are identical [7]. However, they used different assumptions resulting in different predictions of the future. A detailed description of both laws is provided in the extended version of this paper [8].

Depending on the application domain, either Amdahl's or Gustafson's assumptions might be valid, or even something in between. First, we take Amdahl's assumptions to predict the power-constrained performance growth. We assume a symmetric CMP where all cores are being used during the parallel part. The clock frequency of all cores is equal and has been scaled down to meet the power budget. The power-constrained speedup that this symmetric CMP can achieve is given by

$$S_{Amdahl_power-constr._sym.}(t) = \frac{1}{s + \frac{1-s}{N(t)}} \times \frac{f(t)}{f_{orig}} \times \frac{P_{budget}}{P_{total}(t)} \quad (3)$$

and is depicted in Figure 5. We used a serial fraction s ranging from 0.1% to 10%. The figure shows that for $s = 0.1\%$ there is a slight performance drop, compared to ideal, going up to a factor 2 for 2022. However, for $s = 1\%$ there is a performance drop of 10x for 2022 and for $s = 10\%$ there is no performance growth at all.

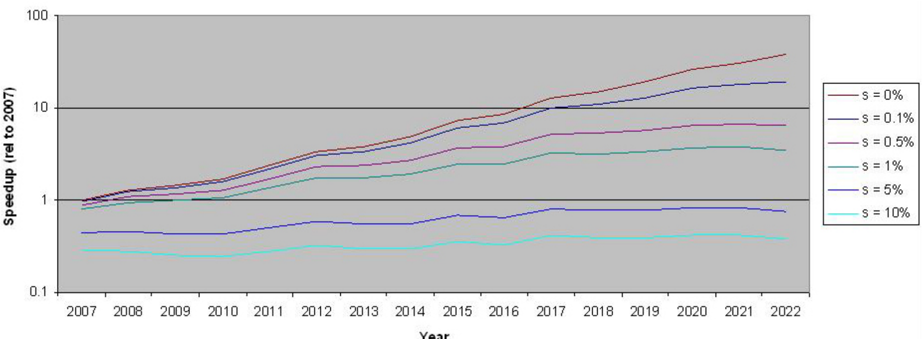


Fig. 5. Prediction of power-constrained performance growth for several fractions of serial code s assuming a symmetric CMP and with Amdahl's assumptions

Indeed we see that Amdahl's prediction is pessimistic, which is an argument for asymmetric or heterogeneous CMPs. If the serial part can be accelerated by deploying one faster core, more speedup through parallelism could be achieved. This one fast core could be an aggressive superscalar core, a domain specific accelerator, or a core that runs at a higher clock frequency than the others. For this experiment we assume identical cores but increase the clock frequency of one core during the serial part. Note that during this time the other cores are inactive and thus the power budget is not exceeded. The speedup this asymmetric CMP can achieve is given by

$$S_{Amdahl_power-constr._asym.}(t) = \frac{1}{s \times \frac{f_{orig}}{f(t)} + \frac{1-s}{N(t)} \times \frac{f_{orig}}{f(t)} \times \frac{P_{total}(t)}{P_{budget}}} \quad (4)$$

and is depicted in Figure 6. Note that both this equation and Equation 3 become identical to Equation 2 if $s = 0\%$. The results show that for $s = 0.1\%$ there is only a very small performance drop compared to ideal parallelization. For $s = 1\%$ the performance drop is 2.3x while for $s = 10\%$ the performance drops 14 times compared to ideal parallelization but considerable performance growth is predicted over time. These results show that asymmetric CMPs are a good choice to improve performance, if Amdahl's assumptions are correct and if the serial fraction is larger than approximately 0.5%.

Second, we predict the power-constrained performance growth using Gustafson's assumptions. Again, we assume a symmetric CMP where all cores are being

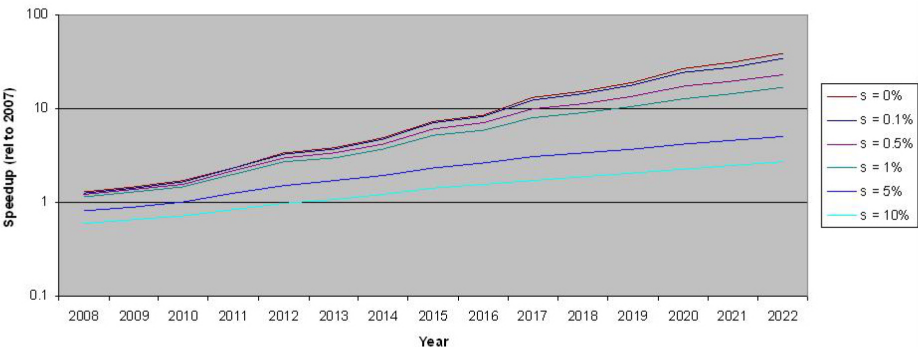


Fig. 6. Prediction of power-constrained performance growth for several fractions of serial code s assuming an asymmetric CMP and with Amdahl's assumptions

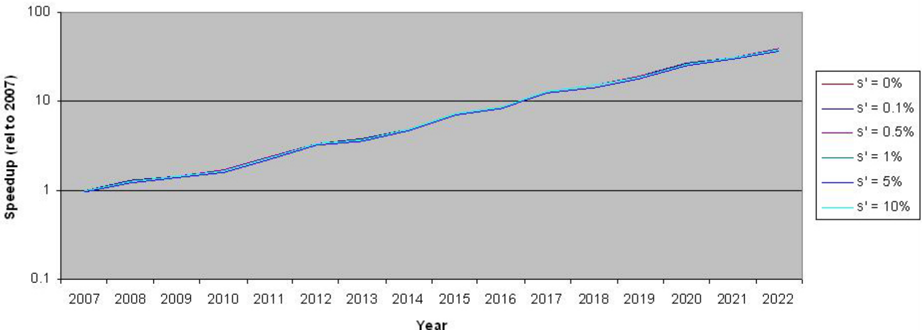


Fig. 7. Prediction of power-constrained performance growth for several fractions of serial code s' with Gustafson's assumptions

used during the parallel part. The clock frequency of all cores is equal and has been scaled down to meet the power budget. The power-constrained speedup that this symmetric CMP can achieve is given by

$$S_{Gustafson_power-constr._sym.}(t) = (N + (1 - N) \times s') \times \frac{f(t)}{f_{orig}} \times \frac{P_{budget}}{P_{total}(t)} \quad (5)$$

and is depicted in Figure 7. The figure shows that for any value s' between 0% and 10% there will be no significant performance loss compared to ideal parallelization. For $s' = 10\%$ in 2022 the performance is 11% less than the ideal parallelization $s' = 0\%$ case. Thus we can conclude that symmetric CMPs are a good choice if Gustafson's assumptions are correct.

3 Conclusions

In this paper we analyzed the impact of the power wall on CMP design. Specifically, we investigated the limits to performance growth of CMPs due to

technology improvements with respect to power constraints. As a case study we modelled a CMP consisting of Alpha 21264 cores, scaled to future technology nodes according to the ITRS roadmap. It was found that in 2022 such a CMP would contain 999 cores, each consuming 1.5 W when running at the maximum possible frequency of 14 GHz. The total CMP, at full blown operation, would consume 1.5 kW while the power budget predicted by the ITRS is 198 W.

From these figures it is clear that power has become a major design constraint, and will remain a major bottleneck for performance growth. However, it does not mean that the power wall has been hit for CMPs. We calculated the power-constrained performance growth and showed that technology improvements enables a doubling of performance every three years for CMPs, which is in line with the number of cores per die.

However, there is another threat for scalable performance which is Amdahl's Law. The speedup achieved by parallelism is limited by the serial fraction s. Using Amdahl's Law we predicted the power-constrained performance growth for several fractions s. For a symmetric CMP, 1% of serial code decreases the speedup by a factor of up to 10. For an asymmetric CMP, where the serial code is executed on a core that runs at a higher clock frequency, 1% of serial code reduces the achievable speedup only by a factor of up to 2.

On the other hand, there is Gustafson's Law which uses different assumptions. The predictions with this Law are much more optimistic as even for $s' = 10\%$ in 2022 the performance loss is only 11% compared to ideal parallelization.

The question whether Amdahl's or Gustafson's assumptions are believed to be valid for a certain application domain is unresolved. Most likely some can be characterized as 'Amdahl', some as 'Gustafson' and other as something in between.

From the results of this paper we conclude that in order to avoid hitting the power wall the following two principles should be followed. First, at the architectural level power efficiency has to become the main design constraint and evaluation criterion. The transistor budget is no longer the limit but the power those transistors consume. Thus performance alone should no longer be the metric but performance per watt (or a similar power efficiency metric like performance per transistor and $BIPS^3/W$). Second, for application domains that follow Amdahl's assumptions asymmetric or heterogeneous designs are necessary. For those the need to speedup serial code remains. A challenge for computer architects is to combine speedup of serial code with power efficiency.

A CMP that follows these principles could for example look like this: a few general purpose high speed cores (e.g. aggressive superscalar), many general purpose power efficient cores (no superscalar, no out-of-order, no deep pipelines, etc.), and domain specific accelerators. The latter provides the most power efficient solution and also allows fast execution of serial code. Furthermore, dynamic voltage/frequency scaling can be applied to optimize the performance-power balance, while hardware support for thread and task management reduces the energy of the overhead introduced by parallelism. A lot more techniques and architectural directions are possible.

Summarizing, from this study we conclude that for the next decade CMPs can provide significant performance improvements without hitting the power wall, even though power severely limits performance growth. Technology improvements will provide the means, however, to achieve the possible performance improvements power efficiency should be the main design criterion at the architectural level.

Acknowledgment

The authors would like to thank Stefanos Kaxiras for his input on the methodology used in this paper.

References

1. Mendelson, A.: How Many Cores are too Many Cores? In: 3rd HiPEAC Industrial Workshop
2. ITRS: International Technology Roadmap for Semiconductors, 2007 Edition (2007), `http://www.itrs.net`
3. Kessler, R.: The Alpha 21264 Microprocessor. Micro 19(2), 24–36 (1999)
4. Stenström, P.: Chip-multiprocessing and Beyond. In: Proc. 12th Int. Symp. on High-Performance Computer Architecture, pp. 109–109 (2006)
5. Hennessy, J., Patterson, D.: Computer Architecture - A Quantative Approach, 4th edn., p. 3. Morgan Kaufman Publishers, San Francisco (2007)
6. Gustafson, J.: Reevaluating Amdahl's law. Communications of the ACM 31(5), 532–533 (1988)
7. Shi, Y.: Reevaluating Amdahl's Law and Gustafson's Law (1996), `http://www.cis.temple.edu/~shi/docs/amdahl/amdahl.html`
8. Meenderinck, C., Juurlink, B.: (When) Will CMPs hit the Power Wall? Technical report, Delft University of Technology (August 2008), `http://ce.et.tudelft.nl/publications.php`

Workshop on Secure, Trusted, Manageable and Controllable Grid Services (SGS 2008)

Grid systems are expected to connect a large number of heterogeneous resources (PCs, databases, HPC clusters, instruments, sensors, visualization tools, etc.), to be used by many users and to execute a large variety of applications (number crunching, data access, multimedia, etc.) and may deal with many scientific fields (health, economy, computing etc.).

Grids are distributed systems and, like them, the notion of security, the way we manage such large system and the way we control the grid system are of particular interest. For instance, the word 'controllable' means how we measure the activity of the grid and how we report it. The word 'manageable' means how we deploy the grid architecture, the grid softwares, and how we start jobs (under controllable events such as the availability of resources). The word 'security' refers to the traditional fields of authentication, fault tolerance but also refers to safe execution (how to certify results, how to adapt computation according to some metric). Moreover, all these services should collaborate, making the building of middleware a challenging problem. In this context questions about who holds the sensitive information, who has permissions to access it, how this information is handled are raised. Therefore, the building of a chain of trust between software components as well as the integration of security and privacy mechanisms across multiple autonomous and/or heterogeneous grid platforms are key challenges.

This workshop aims at gathering papers in the fields of grid/distributed systems and it extends the recent STPG workshop (see CCGRID2008) and the *Journal of Supercomputing* special issue that have served to federate a community of researchers and practitioners. The workshop is also open for contributions in connected fields: P2P systems, sensors networks, networking, large-scale heterogeneous distributed databases.

The Steering Committee invited authors to submit original and unpublished work. We also required that the submission was not being concurrently submitted to another special issue or conference. Papers were reviewed by at least two reviewers. We informed authors that papers without evaluation (either theoretical or experimental) would not be considered.

So, we solicited novel papers on a broad range of topics including but not limited to the following keywords:

- Grid monitoring and controlling (forecasting...) systems
- Grid management systems (deployment of infrastructures and applications)
- Grid security: access control, i.e., authentication, trust-based models
- Infra-structural support for privacy in grid environments: architectures, mechanisms, models, frameworks and implementation
- Specification of grid services for secure components

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 195–197, 2009.
© Springer-Verlag Berlin Heidelberg 2009

- Secure execution and reliability: adaptation, results' certification, safe execution and/or communication
- Aggregation of secure/manageable components into grid middlewares
- Algorithms related to resource brokers, load balancing and heterogeneity
- Applications and experiences with secure/manageable/controllable grid infrastructures

Since the selection of papers of the workshop was not a 'one-shot' process, we requested the authors to carefully read the reviews and to modify the original submission (and the oral presentation) as soon as possible in order to take into account the comments... and the work was done! The selected papers were:

- Attila Kertesz, Ivan Rodero and Francesc Guim Bernat. "Meta-Brokering Solutions for Expanding Grid Middleware Limitations"
- Sebastien Varrette, Jean-Louis Roch, Guillaume Duc and Ronan Keryell. "Building Secure Resources to Ensure Safe Computations in Distributed and Potentially Corrupted Environments"
- Caniou Yves and Jean-Sebastien Gay. "Simbatch: An API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems"
- Heithem Abbes and Jean-Christophe Dubacq. "Analysis of Peer-to-Peer Protocols Performance for Establishing a Decentralized Desktop Grid Middleware"
- Oleg Lodygensky, Gilles Fedak, Gabriel Caillat, Haiwu He and Etienne Urbah. "Towards a Security Model to Bridge Internet Desktop Grids and ServiceGrids"

The workshop started on August 25, 2008, at the Informatics and Mathematics Building. The order for presentations followed the list of selected papers above, and talks were 30 minutes long. The slides for presentation are available online at `http://www.lipn.fr/~cerin/SGS.html`. During the last part of the workshop, we managed a panel discussion in order to isolate main topics of interest for the next workshop and what we think is important for future work.

During the panel session, attendees listed the problems covered in the talks and showed the coherency of the topics of selected papers. At the end, they tried to find a common interest and emerging projects for the future.

For the sake of simplicity, the topics covered during the workshop can be summarized as follows:

1. Analysis of publish/subscribe systems in order to build an institutional / community orchestration tool for desktop grids
2. Meta brokering solutions: how to make grid services more generic
3. Techniques for building secure resources and safe computation
4. An API for simulating and predicting scheduling heuristics
5. Security models for coupling Internet desktop grids and service grids: how to cope with different grid instances according to a secure mode

Attendees concluded that they have all the components to build a result certification service for desktop grids, and the building of such a service introduces challenging efforts in order for it to be:

- Scalable
- Decentralized as much as possible
- Based on standards for cooperation and description
- Based on proved algorithms for the certification algorithms
- With performance
- Interoperable with other grids flavors and with a secure mode to grant operations (delegating rights)
- Able to be simulated in order to validate the component

From the certification side, they noticed that a large set of techniques exist, among them replication, partial replication of randomly selected tasks, lists of repudiation to minimize re-computations, and quiz. From the meta-brokering point of view, they noticed that standards are now available for the description scheduling languages, for the capability description languages, for global job identifiers, for security management so that the certification services could be build on top of theses facilities.

Finally, attendees recalled the questions of "what is the best institutional form / organization able to support such work" and how to extend the topics of the workshop for the 2009 edition.

October 2008 Christophe Cérin

Meta-Brokering Solutions for Expanding Grid Middleware Limitations

Attila Kertész[1], Ivan Rodero[2], and Francesc Guim[2]

[1] MTA SZTAKI Computer and Automation Research Institute
H-1518 Budapest, P. O. Box 63, Hungary
attila.kertesz@sztaki.hu

[2] Technical University of Catalonia (UPC)
Jordi Girona 29, 08034 Barcelona, Spain
{irodero,fguim}@ac.upc.edu

Abstract. Grid Resource Management tools evolved from manual discovery and task submission to sophisticated brokering solutions. User requirements created certain properties that resource managers have learned to support. This development is still continuing, and users already need to stress themselves to distinguish brokers and migrate their applications when they move to a different grid. Moreover, grid interoperability have emerged the need for higher level brokering services. This paper introduces a meta-brokering approach that means a higher level resource management by enabling automatic and simultaneous utilization of Grid Brokers. First we gather the requirements of this novel middleware service then define a general meta-brokering architecture. Finally we show how meta-brokers can be implemented in different grid environments and we conclude with their evaluations.

Keywords: Grid Interoperability, Grid Meta-brokering, Grid Scheduling, Grid Services.

1 Introduction

In the last decade grid resource management has evolved from manual discovery and task submission to sophisticated brokering solutions. Nowadays research directions are focusing more on user needs: more efficient utilization and interoperability play the key roles. Grid resource management is probably the research field most affected by user demands. Though well-designed, evaluated and widely used resource brokers, meta-schedulers have been developed, new capabilities are still required, such as agreement and interoperability support. These two directions also depend on other grid middleware capabilities and services, and since they cannot cross the border of these middleware solutions, they need significant changes affecting the whole system. Trying to enlarge the limitation borders, in this paper we introduce a meta-brokering approach as an evolutionary step for grid resource management, which does not need radical changes to the whole system. The need for interoperability among different grid systems has raised

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 199–210, 2009.
© Springer-Verlag Berlin Heidelberg 2009

several questions and directions. The advance of grids seems to follow the way envisaged and assigned by the Next Generation Grids Expert Group [1]. The SOKU architecture [1] enables more flexibility, adaptability and advanced interfaces, therefore interoperability is evident and congenital in these systems. Moving towards this direction two works envisaged WWG (World Wide Grid) as the next generation of grids in a similar manner as the Internet was born. One of the first vision is the InterGrid [2], which promotes interlinking of grid systems through peering agreements to enable inter-grid resource sharing. This approach is a more economical view, where business application support is a primal goal, and this also supposed to establish sustainability. The second approach is detailed in [3] which states that three major steps are required to create the WWG architecture: existing production grids should be connected by uniform meta-brokers, workflow-oriented portals should interface these meta-brokers, and a repository of workflow grid services should be created. As a final solution the inter-connected meta-brokers should solve the interoperability among different grids through uniform interfaces. Both visions proceed from the current grid architectures and conclude in a more or less redesigned one.

In this paper we gather the requirements of this novel middleware service that define a general meta-brokering architecture. Moreover, we show how meta-brokers can be implemented in different grid environments. Finally, we evaluate our implementations in two diferent simulation environment, and demonstrate that the meta-brokering approach can reduce execution time and improve the overall interoperable system performance. The main contributions of this work are the design, implementation, and evaluation of the meta-brokering approach in two different scenarios.

The rest of the paper is organized as follows. In Section 2 we present the related work in higher level brokering directions. In Section 3, we introduce a general meta-brokering architecture by gathering its requirements. In Section 4, we show how the presented architecture can be realized in two different grid user environments, and we describe the simulation environments used for evaluating these realized solutions. Finally, in Section 5, we present the conclusions and future work.

2 Related Work

Several research groups have noticed interoperability problems at the level of grid resource management. One of the promising approaches aimed at enabling communication among existing resource brokers. The GSA-RG of OGF [15] is currently working on a project enabling grid scheduler interaction. They plan to define common protocol and interface among schedulers enabling inter-grid usage. In this work they propose a Scheduling Description Language to extend the currently available job descriptions. Another instance of this approach is the Latin American Grid (LA Grid) initiative. The meta-scheduling project in LA Grid [16] aims to support grid applications with resources located and managed in different domains. They define meta-broker instances with a set of functional

modules: connection management, resource management, job management and notification management. These modules implement the APIs and protocols used in LA Grid through web services. Each meta-broker instance collects resource information from the other instances having a view of all resources in aggregated form. The Koala grid scheduler [4] was designed to work on DAS-2 interacting with Globus middleware services with the main features of data and processor co-allocation, lately it is being extended to support DAS-3 and Grid'5000. To inter-connect different grids, they have also decided to use inter-broker communication. Their policy is to use a remote grid only if the local one is saturated. In an ongoing experiment they use a so-called delegated matchmaking (DMM), where Koala instances delegate resource information in a peer-2-peer manner. Gridway introduces a Scheduling Architectures Taxonomy [5], where they describe a Multiple Grid Infrastructure. It consists of different categories, we are interested in the Multiple Meta-Scheduler Layers, where Gridway instances can communicate and interact through grid gateways. These instances can access resources belonging to different administrative domains (grids/VOs). The basic idea is to pass user requests to another domain when the current is overloaded. This approach follows the same idea as the previously introduced DMM.

Comparing the previous approaches, we can see that all of them use a new method to expand current grid resource management boundaries. Meta-brokering appears in a sense that different domains are being examined as a whole, but they rather delegate resource information among domains, broker instances or gateways. Usually the local domain is preferenced, and when a job is passed to somewhere else, the result should be passed back to the initial point. In the next section we focus on a solution that utilizes and delegates broker information by reaching the brokers through their current interfaces.

3 General Meta-Brokering Architecture

We have developed solutions to make resource managers' data available for cooperated, automatic processing in the form of metadata. We provide language schemas to store and share this metadata, and to be processed by various scheduling policies. Using these advanced techniques we present a meta-brokering architecture that enables a higher level brokering that has a global, up-to-date view of broker capabilities and availability. Figure 1 is intended to show all the components and tools needed by a General Meta-Broker. In the following we describe these components by introducing the main requirements of this higher level brokering service:

- **JDL – Job Description Language:** Since the goal of the meta-brokering service is to offer a uniform way to access various grids, a unified description format is needed to specify all the user requirements.
- **CDL – Capability Description Language:** Each broker has a different set of functionalities, they can be specialized in different application-types. In order to store and track these properties, it is required to use a CDL.

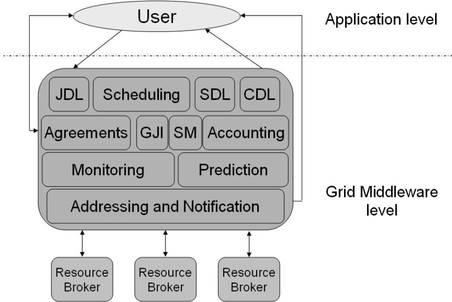


Fig. 1. Required components and tools of a General Meta-Broker

It should be general enough to include all the service capabilities (interfaces, job submission, monitoring and agreements).

- **SDL – Scheduling Description Language:** Besides CDL and JDL the scheduling requirements and policies also need to be stored separately. The users can express their needs by extending the JDL with SDL, and the scheduling policies and methods of the brokers can be stored in this format.
- **Scheduling:** This component performs the scheduling (matchmaking) of incoming user requests. A proper grid broker (which implies a domain, VO or execution environment) needs to be selected for a user job taking into account the available scheduling policies.
- **GJI – Global Job Identifiers:** It is important to have unique mapping of user jobs to different grids. An implementation can be a single job ID provider for the meta-brokering system or simply using each broker system as a prefix for the assigned grid job ID.
- **SM – Security Management:** The role of this component is to provide secure access to the interconnected domains. For example, different user certificates, proxies may be accepted in different VOs and grids. Providing a transparent way for users these various proxies also need to be handled by the Meta-Broker. (In most cases we rely on the underlying Grid Security Infrastructure (using proxies).)
- **Accounting Mechanism:** The GJI and SM can be a part of a global accounting component. The role of this mechanism is to manage user access by pre-defined policies. Though grid economy is still in a pre-mature state, in the future the meta-brokering service should also serve business grids.
- **Agreements Mechanism:** This component is in connection with the Accounting mechanism. Service Level Agreements (SLA) are planned to be used in future generation grids. The role of this part is to negotiate user requirements, which can also affect scheduling policies. When agreements will be generally accepted and used, this mechanism should be extended to do negotiations with the brokers.
- **Monitoring Mechanism:** Reliable operation requires global monitoring, in terms of the inter-connected brokers, reachable domain, grid resources and

loads, and local component functionalities. Self-awareness and fault tolerance need to be provided by the system itself, which needs extensive monitoring.
- **Prediction Mechanism:** This component is in connection with the Monitoring and Scheduling mechanisms. It is necessary to perform calculations of broker availability, service utilization and user request load to cope with the highly dynamic nature of grids.
- **Addressing and Notification Mechanism:** This component is responsible for accessing the inter-connected resource brokers, and managing communication including local events and external job state notifications.

In addition to the architecture model and the required interfaces it is important to deal with scheduling at the meta-brokering level. Based on the components of the architecture shown in Figure 1, we propose a general meta-brokering policy relying on the capabilities and measured performances of the utilized brokers. The selection of the appropriate brokering system can be done using a multi-criteria algorithm that can take into account the gathered and predicted metadata stored in SDL and CDL. The general scheduling policy examines the job description (JDL) and matches its requirements to the metadata stored in CDL and SDL. The matchmaking algorithm consists of the following steps: In the first phase the basic job requirement attributes and the scheduling constraints are matched against the basic broker capabilities: this selection determines a group of brokers that are able to fulfill the job request. In this phase those brokers are kept that are able to submit the specified job type and work with the requested middleware services. In the second phase the brokers are filtered by all the scheduling requirements of the user job. For all of the brokers that could pass these filtering phases a rank is computed by the actual broker performance and domain load. For the brokers that could only pass the first phase reduced ranks are assigned. Finally the list of the brokers is ordered by these ranks. The broker with the highest rank is selected for submission. Different scheduling policies can be defined regarding the data stored in SDL. It also depends on the deployed architecture, which policies are implemented. In economy-based grid environments scheduling policies could rely on the agreements and accounting information to match user budget and maximize domain earnings. An ongoing work in UPC investigate to use policies based on historical job/resource information to make scheduling decisions. Later we can use the achievements of this approach to improve the meta-brokering algorithm and create another policies with estimation and prediction results.

4 Meta-Broker Architecture Realizations

Having all the requirements defined, we are able to implement and build a meta-brokering service. As we stated in the previous section, semantics are crucial to establish interoperability. Standardization should be taken into account during the design and development of sustainable solutions. To walk on this way, we use the standards and widespread technologies, where applicable. To describe

user jobs we use JSDL (Job Submission Description Language) [17] and WS-Agreement for handling agreements. Regarding CDL we developed a language called BPDL (Broker Property Description Language) [8], which had also incorporated SDL. To provide full support for our meta-brokering approach we have revised and modified BPDL and gathered the scheduling-related attributes to MBSDL (Meta-Broker Scheduling Description Language). The structure of the new BPDL – that we call BPDL 2.0 –, remains nearly the same, we have only clarified some attributes, added missing ones and separated the scheduling-related ones to MBSDL. The updated BPDL version includes: *BrokerID*, *Interface* (to providee metadata about the accessibility and notification), *Monitoring*, *Security*, and *Middleware* (kind of middleware, grid or VO the broker can operate), *JobType*, *RemoteFileAccess*. We use the *PerformanceMetrics* field to store how successfully the broker performed job requests, and how reliable its services are. The Prediction attribute can be used to store predicted data about broker availability and reliability. The MBSDL language can be used to extend BPDL with scheduling related attributes. Since JSDL is also lacking these attributes, MBSDL can also be regarded as a JSDL extension. Its schema contains three fields: *Constraints* (that have to be fulfilled during scheduling), *QoS (Quality of Service) requirements*, and *Policy* (for scheduling policies). Taking into account these ideas, in the following, we present two different realizations of the presented meta-brokering approach. The first one is the Grid Meta-Broker that has been evaluated in a GridSim-based environment, and the second one is the extension of eNanos that has been evaluated with the Alvio simulation framework.

4.1 The First Solution: Grid Meta-Broker

The first realization, the Grid Meta-Broker [7], will be used to solve meta-brokering in the future version of P-GRADE Portal [6]. Nevertheless it has been designed as a standalone, standards-based Web Service, therefore it is grid middleware and portal independent. It has been derived from the architecture shown in Figure 1. This service consists of the following components (Figure 2): the Translator is responsible for translating the JSDL of the user job to the language of the appropriate broker that the Grid Meta-Broker selects for submission. The users can use the MBSDL to specify scheduling related attributes, and the Meta-Broker uses BPDL together with MBSDL to store broker properties. The Information Collector (IC) component stores the data of the reachable brokers and historical data of the previous submissions in BPDL. The load of the resources behind the brokers is also taken into account to help the MatchMaker component to select the proper environment for the submitted job. There are IS Agents reporting to the IC, which regularly check the load of the underlying grids of each connected resource broker. The previously introduced language attributes are used for matching the user requests to the description of the interconnected brokers: which is the role of the MatchMaker component. The Invokers are broker-specific components: they communicate with the interconnected brokers, invoking them with job requests and collecting the results. In the JSDL extension, the middleware constraint field can be used to specify

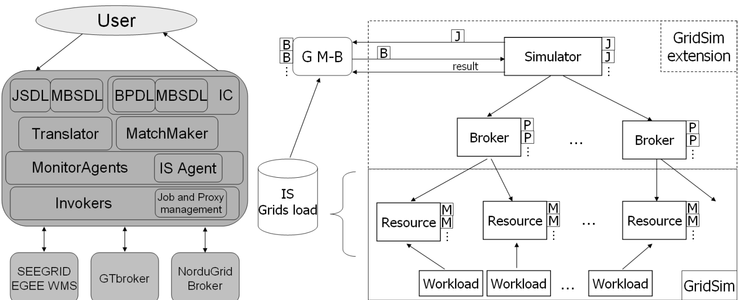
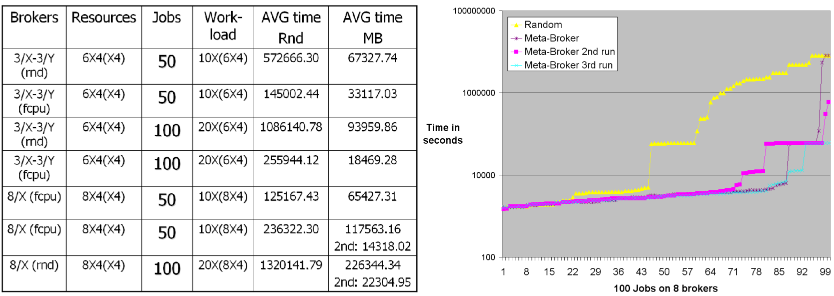


Fig. 2. Grid Meta-Broker architecture (on the left), and its simulation architecture with GridSim (on the right)

proxy names for grids/VOs. This information is used by the Invokers to select the valid proxy from the uploaded files for the actual job submission.

We have created a prototype that is mature enough to perform evaluation test in a simulated environment. We have chosen the GridSim toolkit [9] for our simulations, because it is one of the most widely used and accepted grid simulation tools. It can be used for simulating and evaluating VO-based resource allocation, workflow scheduling, and dynamic resource provisioning techniques in global Grids. It supports modeling and simulation of heterogeneous Grid resources, users, applications, brokers and schedulers in a Grid computing environment. It provides primitives for the creation of jobs (called gridlets), mapping of these jobs to resources, and their management, therefore resource schedulers can be simulated to study scheduling algorithms. GridSim provides a multilayered design architecture based on SimJava [10], a general purpose discrete-event simulation package implemented in Java. It is used for handling the interaction or events among GridSim components. All components in GridSim communicate with each other through message passing operations defined by SimJava.

Our general simulation architecture can be seen in the right part of Figure 2. On the right-bottom part we can see the GridSim components used for the simulated grid systems. Resources can be defined with different grid-types. Resources consist of more machines, to which workloads can be set. On top of this simulated grid infrastructure we can set up brokers. Brokers are extended GridUser entities: they can be connected to one or more resources, they report to the IS Grid load database (which has a similar purpose as a grid Information System), different properties can be set to these brokers (agreement handling, co-allocation, advance reservation, etc.), properties can be marked as unreliable, different scheduling policies can be set for each broker, and resubmission is used, when a job fails due to resource failure. The Simulator (User in the real world) is an extended GridSim entity: it can generate a requested number of gridlets (jobs) with different properties, start and run time (length). It is connected to the brokers and able to submit jobs to them, the default job distribution is the random broker selection (though at least the middleware types are taken into



Brokers	Resources	Jobs	Work-load	AVG time Rnd	AVG time MB
3/X-3/Y (rnd)	6X4(X4)	50	10X(6X4)	572666.30	67327.74
3/X-3/Y (fcpu)	6X4(X4)	50	10X(6X4)	145002.44	33117.03
3/X-3/Y (rnd)	6X4(X4)	100	20X(6X4)	1086140.78	93959.86
3/X-3/Y (fcpu)	6X4(X4)	100	20X(6X4)	255944.12	18469.28
8/X (fcpu)	8X4(X4)	50	10X(8X4)	125167.43	65427.31
8/X (fcpu)	8X4(X4)	50	10X(8X4)	236322.30	117563.16 2nd: 14318.02
8/X (rnd)	8X4(X4)	100	20X(8X4)	1320141.79	226344.34 2nd: 22304.95

Fig. 3. Grid Meta-Broker evaluation results

account). In case of job failures a different broker is selected for job submission. It is also connected to the Grid Meta-Broker (denoted with G M-B in the figure) through its web service interface and able to call its matchmaking service. Before the Meta-Broker is used for broker selection, it have to be configured with BPDL descriptions, and the job requests need to be submitted in JSDL.

The first evaluation environment consisted of 6 brokers operating on four resources each. Each resource had four machines. Three brokers used resources with GRID_X middleware and the other three used resources with GRID_Y middleware. Four properties were distributed among the brokers, each broker had one unreliable property (50% failure) (this usually happens in real Grids). Half of the jobs were set to GRID_X, the rest to GRID_Y. Each job had only one property and all the four properties were distributed equally among the jobs. The run time of the jobs took around 5 minutes each. 20 percent of the jobs had no special property, the rest were assigned to two properties out of the four (they were distributed equally). We evaluated four different scenarios, Figure 3 shows a table with their parameters. The first line means all the brokers used random selection policy, 50 jobs were submitted into the system and 10 jobs were submitted to each resource as background workload. We used the cleaned SDSC Blue Horizon workload logs from the Parallel Workloads Archive [13]. The fifth column shows the average simulation run time of the jobs, when we used random distribution among the brokers. The sixth column shows the average simulation run time for the jobs submitted to brokers selected by the Meta-Broker. Due to the broker property distribution in the first simulation setup, a job with a special property running on either middleware could surely successfully run only on one specific broker. This caused overloading of some brokers even with the use of the Meta-Broker, therefore we created a different environment. The second time we set up 8 brokers operating again on 4-4 resources, but all having the same GRID_X middleware. The same property distribution was used for the jobs. The brokers had 2-2 special properties again, but every second broker had one unreliable property (in this case two brokers could run some job the same time without any failures). In Figure 3 the last two lines contain additional information in their last columns. This means we repeated the measurement again, in a way that the brokers were aware of previous broker failures. In these

cases the first run can be regarded as a teaching phase, therefore we measured better performance.

On the right of Figure 3, we highlighted a diagram of the last evaluation phase, which best demonstrates the difference of simulation run time with and without the use of the Meta-Broker. The results show that the Grid Meta-Broker provides less execution time by automating broker and grid selection for users. During utilization it is able to adapt to broker failures and to avoid selecting overloaded grids. Future work aims at enhancing its matchmaking algorithm and introducing teaching phases for better adaptation.

4.2 The Second Solution: Meta-Brokering in HPC Centers with eNANOS

In our past research we have implemented a brokering approach within the eNANOS Resource Broker [11]. It performs the job scheduling based on FCFS and the resource selection based on the matchmaking approach. It also uses hard user criterion (requirements) and soft ones (recommendations) with a predefined set of priorities for computing ranks (the resource with the highest rank is selected for submission). Recently, this scheduling policy has been modelized in the broker layer of Alvio. The Alvio Simulator is a C++ event driven simulator that has been designed and developed to evaluate scheduling policies in high performance architectures. It provides ways to evaluate environments from local centers to meta-brokering solutions. Regarding the local architecture models, like other simulators, given a workload and an architecture definition, it is able to simulate how the jobs would be scheduled using a specific scheduling policy (such as FCFS, or Backfilling policies). The main contribution of this simulator at this level is that it does not only allow modelling the job workflow in the system, but also allows simulating different resource allocation policies.

The other contribution of this simulator is the modelling of resource usage with different jobs running in the system. The researcher is able to specify the setup of the resources available in the architecture (eg. the amount of memory bandwidth in each node) and the amount of resources to be used by the workload jobs. In scenarios for distributed architectures composed by several HPC centers, the simulator allows to simulate multi-site systems (also known as brokering system) and meta-brokering systems. Figure 4 presents the different components that are instantiated inside the simulator in the meta-brokering scenario. The meta-system entity is a conceptual component that models the different elements that are included in the meta-environment. It can contain a centralized meta-broker scheduling entity which potentially can implement centralized-based policies (which is illustrated in Figure 4). It also allows to evaluate P2P meta-brokering strategies. This meta-system component contains a set of different virtual organization (VO) elements. Each of them contains a global broker entity that is responsible for managing the jobs that users submit to the VO. Furthermore, it contains a set of centers that model the typical HPC local resources introduced previously. Thus, the different centers of the local-scenarios of the simulation model are also instantiated. For each of the

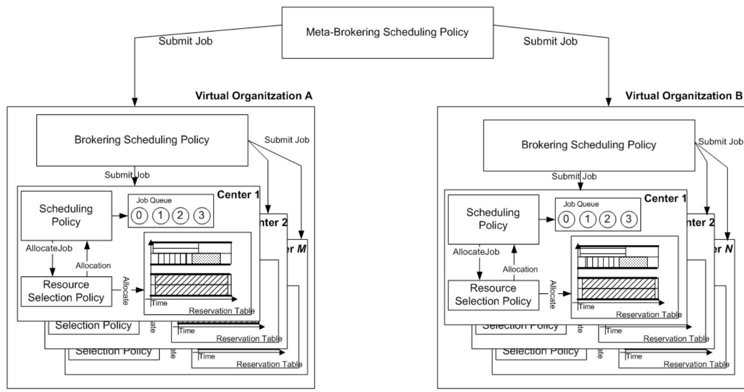


Fig. 4. Meta-brokering Model with a Centralized Approach

center components a local scheduling policy, a resource selection policy and the reservation table are created.

Similarly, we have proposed and implemented a prototype of the *BestBrokerRankPolicy*. It basically selects the most appropiate broker to submit a job based on a set of rank values corresponding to the different brokers rather than using the local resource information. But for obtaining these values it can consider resource information in an aggregated form and apply an impact factor to each attribute. Since we consider forwarding jobs between brokers, we also consider the average slowdown of the brokers as a QoS metric. An example of the resource aggregated form can be found in [12] and the policy will be widely described and evaluated in a separate paper. In the evaluation we have used traces from the Grid Workloads Archive [14]. In particular we have used the workloads from the DAS-2, Grid5000, and Sharcnet systems. We have selected some weeks of each workload trace (around 10.000 jobs in total) with a maximum demand of 512 CPUs and avoiding failed jobs. We found that the average execution time of the jobs depends on the workload, but in average a simulation took around 2 hours. We have defined 3 simulation environments, one for each workload with its own broker instance. For the DAS-2 system we modelized 5 resources with a total of 336 CPUs, 10 resources with a total of 574 CPUs for the Grid5000 system, and 10 resources with a total of 3200 CPUs for the Sharcnet system. We have reduced the inter-arrival times for the jobs by a factor of two. This has allowed us to increase the pressure to the system by incrementing the load. In the evaluation we have simulated four different scenarios. First, we simulated the scenario where each virtual organization schedules only their own jobs. Thus we evaluated the performance of each of the different virtual organizations without any connection among them. For the second time, we simulated interconnected VOs using the same workloads. Thereby, we have evaluated the effect of the job forwarding between the different grid brokers in a P2P fashion.

Table 1 shows the performance results for the four different scenarios. We present the total workload execution time, and the 95th percentile and average

Table 1. Evaluation results with Alvio

Workload	Exec Time (h)	Avg Wait (s)	Avg SLD	CPU Util
DAS-2	1,486	1,789	11,26	28 (8,33%)
Grid5000	1,565	4,850	13,06	39 (6,79%)
Sharcnet	1,596	3,011	15,78	200 (6,25%)
Overall	1,596	3,216	13,36	267 (7,12%)
Meta-brokering	1,410	274	1,34	314 (7,63%)

metrics for the wait time, slowdown and CPU utilization. The first three rows of the table show the performance values for the three VOs. In all the three scenarios we can notice that the processor utilization is relatively small. The Grid5000 and Scharnet show similar values in terms of execution time and CPU utilization. The DAS-2 shows substantially lower execution time, wait time, and slowdown. The third row of the table shows the average of all the variables considering all the three VOs. In the last row of the table, that contains the meta-brokering approach, the system performance is around 11%, which is better then considering the job forwarding. When the P2P meta-brokering is enabled the average wait time is reduced by a factor of 11 times. Futhermore, the slowdown is also reduced by a factor of 10 times. Thus these results indicate that this approach potentially can provide good performance in large scenarios.

5 Conclusions and Future Work

In this paper we have shown the ongoing research directions in the field of grid resource management and stated the necessary steps to be taken to establish a higher level of grid interoperability with meta-brokering by expanding current middleware limitations. We also have defined the essential requirements of a novel grid middleware service called Meta-Broker, we have proposed a general architecture, and we have shown how it can be realized in two different grid environments. Finally, we have developed simulators to these environments to evaluate our implemented prototypes with different workload samples. The results of our simulation-based evaluations show that our meta-brokering approach has a significant gain compared to the approach based on independent brokering systems. In particular, we have measured less total job execution times and identified significant improvement of the overall interoperable systems performance. Our future work aims at investigating new meta-brokering scheduling policies with SLA usage and enlarging the simulation scenarios to evaluate grid environments with more brokers. We also plan to validate our approach in real execution environments such as the P-GRADE portal and the LA Grid infrastructure.

Acknowledgement

This research work was partly supported by the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265) and by the Spanish Ministry of Science and Technology under contract TIN2007-60625.

References

1. Next Generation Grids Report: Future for European Grids: GRIDs and Service Oriented Knowledge Utilities – Vision and Research Directions 2010 and Beyond (NGG3) (December 2006)
2. Assuncao, M.D., Buyya, R., Venugopal, S.: InterGrid: A Case for Internetworking Islands of Grids. In: Concurrency and Computation: Practice and Experience (CCPE), July 2007. Wiley Press, New York (2007)
3. Kacsuk, P., Kiss, T.: Towards a scientific workflow-oriented computational World Wide Grid. CoreGRID Technical Report TR-115 (December 2007)
4. Iosup, A., Epema, D.H.J., Tannenbaum, T., Farrellee, M., Livny, M.: Inter-Operating Grids through Delegated MatchMaking. In: Supercomputing 2007 (SC 2007), Reno, Nevada (November 2007)
5. Vazquez, T., Huedo, E., Montero, R.S., Llorente, I.M.: Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 372–381. Springer, Heidelberg (2007)
6. Kacsuk, P., Sipos, G.: Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal. Journal of Grid Computing, 1–18 (Feburary 2006)
7. Kertesz, A., Kacsuk, P.: Meta-Broker for Future Generation Grids: A new approach for a high-level interoperable resource management. In: CoreGRID Workshop on Grid Middleware in conjunction with ISC 2007 conference, Dresden, Germany (June 2007)
8. Kertesz, A., Rodero, I., Guim, F.: Data Model for Describing Grid Resource Broker Capabilities. In: CoreGRID Workshop on Grid Middleware in conjunction with ISC 2007 conference, Dresden, Germany, June 25-26 (2007)
9. Buyya, R., Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency & Computation: Prac. & Exp. (November-December 2002)
10. Howell, F., McNab, R.: SimJava: A discrete event simulation library for Java. In: Intl. Conference on Web-Based Modeling and Simulation, San Diego, USA (1998)
11. Rodero, I., Corbalán, J., Badía, R.M., Labarta, J.: eNANOS grid resource broker. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 111–121. Springer, Heidelberg (2005)
12. Bobroff, N., Fong, L., Kalayci, S., Martinez, J.C., Rodero, I., Sadjadi, S.M., Villegas, D.: Enabling Interoperability among Meta-Schedulers. In: 8th IEEE International Symposium on Cluster Computing and the Grid, Lyon, France (2008)
13. Parallel Workloads Archive, `http://www.cs.huji.ac.il/labs/parallel/workload/`
14. Grid Workloads Archive, `http://gwa.ewi.tudelft.nl/`
15. `https://forge.gridforum.org/sf/projects/gsa-rg`
16. `http://latinamericangrid.org/`
17. `http://www.ogf.org/documents/GFD.56.pdf`

Building Secure Resources to Ensure Safe Computations in Distributed and Potentially Corrupted Environments

Sebastien Varrette[1,2], Jean-Louis Roch[2], Guillaume Duc[3,4], and Ronan Keryell[3,4,5]

[1] University of Luxembourg, CSC Research unit, Luxembourg
[2] MOAIS/SAFESCALE Project, CNRS–INRIA, LIG Laboratory, Grenoble, France
[3] Institut TÉLÉCOM; TÉLÉCOM Bretagne; HPCAS/SAFESCALE group, Computer Science Laboratory, Plouzané, France
[4] Université Européenne de Bretagne, France
[5] HPC Project, Meudon, France

Abstract. Security and fault-tolerance is a major issue for intensive parallel computing in pervasive environments with hardware errors or malicious acts that may alter the result. In [1,2] is presented a novel, robust and secure architecture able to offer intensive parallel computing in environments where resources may be corrupted. Some efficient result-checking mechanisms are used to certify the results of an execution. The architecture is based on a limited number of safe resources that host the checkpoint server (used to store the graph) and the verifiers able to securely re-execute piece of tasks in a trusted way.

This article focus on the effective construction of strongly secured resources. Our approach combine both software and hardware components to cover the full spectrum of security constraints. The proposed computing platform is validated over a medical application and some experimental results are presented.

1 Introduction

Nowadays, intensive parallel applications require often global computing platforms composed of scattered resources that are interconnected through the Internet. Such platforms, called *grids* [3], are more and more used since the 90's.

As any open infrastructure using a public network, such an infrastructure is the target for various threats, more precisely scans, Denial-of-Services (DoS), intrusions, applicative vulnerabilities and malwares (worms, virus, trojan horses). Grid computing history revealed also selfish behavior as the incentives proposed to the users attract cheaters who seek to obtain these rewards with little or no contribution to the system. The cheating causes are manifold and sometimes not even malicious. Yet the consequences can be modeled by intermediate results tampering. For instance, this issue has been first illustrated in Seti@Home [4] where a modified client were designed claiming it improves FFT computation

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 211–222, 2009.
© Springer-Verlag Berlin Heidelberg 2009

but rather introduced rounding errors that canceled months of world-wide computation. To sum up, large scale computing systems raise various concerns in terms of security and data privacy. In particular:

- users and resources should be authenticated;
- only authorized users should be allowed to access and to use only the resources of the grid allocated for their own purpose;
- communications should be ciphered to ensure privacy but also integrity;
- data should be securely stored;
- the system should remain operative even in case of failure of some grid components or disconnections which are relatively frequent events. In other words, the integrity of the execution should be guaranteed;
- the resources of the grid should be protected from malicious code;
- the infrastructure should not fail even if some parts of the grid are under control of a pirate or, worse, a wicked system administrator, since, on quite large grids, it is no longer possible to know neither to trust every remote system administrator or computer owner.

Other constraints could intervene such as the mandatory interaction between global and local security policies or the required Single Sign On (SSO) for users authentication on the resources. In this article, we describe a computing platform able to address these issues and propose its validation on a concrete medical application. More precisely, section 2 details the secured large scale computing platform used inside the SAFESCALE project to ensure the computation resilience despite crash faults and malware attacks that lead to computation alterations. Our infrastructure assumes the availability of strongly secured resources which run processes in a trusted and trustworthy way. Section 3 presents the core of this paper with the construction of such secured resources. Our talk is mainly oriented toward open-source solutions in computing grids based on Unix or Linux operating systems which constitute major actors in this field. While this building issue is generally treated with software only solutions, our approach combines both software and hardware elements (*i.e.* Cryptopage secure processors) to deal with the full spectrum of security constraints. Finally, section 4 describes the validation of the proposed platform within a concrete medical application. Our experimentations demonstrates the very low overhead induced by the strongly secured resources described in this article.

2 The SAFESCALE Computing Platform

This section describes the SAFESCALE computing platform by first reminding the general guidelines leading to secure computing grid.

A prerequisite would be to rely on safe resources. At this level, a few basic protections could be applied, more precisely:

- the control of user rights, together with the limitation of available services and the enforcing of quotas (either on I/O or processes);

- an up-to-date system with an network firewall to limit and control the network traffic, the opened ports and the way a dedicated service is launched;
- monitoring and audit tools such as anti-virus/spyware software to control and report file change(s) and processes on the systems. This helps to detect corruption attempts on the system integrity;
- the use of sandbox environments *i.e.* confined execution environments, which could be used to run untrusted programs. The limitation could intervene either at the level of the file system hierarchy seen by the executed process (with tools such as `chroot` or BSD `jail`) or with runtime sandboxes through virtual machines. The applications could also contain a sandbox mechanism within themselves. Proof-carrying code (PCC) [5] is such a technique used for safe execution of untrusted code. The basic idea is to require the code producer to generate a formal proof that the code meets the safety requirements set by the code receiver.

This list is not exhaustive. Additionally, there is no need to develop too much this aspect as this degree of control over the computing resources is far from being guaranteed on large scale computing grids. We now focus on the generic security concerns to be addressed, namely confidentiality, authentication, access control and integrity.

Confidentiality. As for communication privacy, it is sufficient to rely on the encrypted tunnels provided by Virtual Private Network (VPN), SSH (as used on the SAFESCALE platform described in this paper), Ipsec or SSL, the later being used in the Globus middleware.

As regards the confidentiality of the executed code, two approaches can be generally distinguished. The first one relies on encrypted computations [6] yet with poor efficiency and limited functionality [7]. The other one consists in various code transformation such as obfuscation [8]. The code is then transformed at the source or binary level to render more complex its readability and also change the data coding. Of course, often this slows down the execution and it is not impossible for someone very motivated to reverse-engineer the obfuscating process since the code is available for execution and can be exercised at will. Another alternative is to transform an algorithm that produces data from input data into an isomorphic one that acts on ciphered input and produces ciphered output. It is also possible to add some authentication mechanisms in it to certify the computation done. For some simple algorithms there exist such efficient isomorphic algorithms but, unfortunately, this seducing approach has an intractable complexity for real life programs [9]. Since basic computations (as, for example, floating point operations that are highly optimized in modern processors), are transformed in elementary operations executed to emulate some enciphered circuits, the expected efficiency on grids is no longer possible. In all cases, as this last domain requires further studies, our platform does not yet support this functionality.

Authentication & access control. Designing a robust authentication system in distributed environments has been extensively studied [10,11,3].

Efficient solutions depend on the grid topology. In multi-clusters environments, a Kerberos [12] approach should be privileged. On grids of clusters topology, the authors of [10] demonstrate an adapted and efficient solution based on LDAP servers that broadcast authentication information. The SAFESCALE platform typically relies on this authentication system. Finally, for grids of bigger size, the Globus [13] middleware is probably the most adapted, more precisely through the GSI module (Grid Security Infrastructure) which depends on a PKI (Public Key Infrastructure) to establish certification chains between the entities of the grid (either users or services).

Integrity. Resilience in grid execution is a prerequisite that should be embedded in the application: at this scale, component failures, disconnections or results modifications are unfortunately part of operations, and applications have to deal directly with repeated failures during program runs.

In [1,2], the authors present a robust and secure architecture able to deal with intensive parallel computing in environments where resources could be corrupted. The corruption could be caused by DDoS attacks, virus or trojan horses, as expounded in the precedent section. The proposed approach uses a portable representation of the distributed execution: a bipartite Direct Acyclic Graph $G = (\mathcal{V}, \mathcal{E})$. The first class of vertices is associated to the tasks (in the sequential scheduling sense) whereas the second one represents the parameters of the tasks (either inputs or outputs according to the direction of the edge). Such a graph is illustrated in Figure 1.

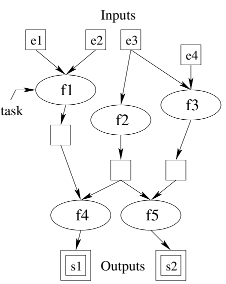


Fig. 1. A data-flow graph with five tasks

Using this representation, portable fault-tolerance mechanisms for heterogeneous multi-threaded applications have been proposed [14]. Since we have a clean separation of the applications in tasks that communicate only through their defined inputs and outputs with no other side effect, each one can be run again in case of trouble. Furthermore, efficient result-checking mechanisms exploiting the graph have been developed [1,15] and are able to assert the behaviour and the results of an execution. In particular, the $EMCT$ certification algorithm will be considered in the sequel as it induces a low overhead for the probabilistic certification of programs composed by either independent, recursive or divide-and-conquer tasks [16], even if the error probability ϵ of $EMCT$ is customized to a very low value (10^{-6} for instance).

The proposed approaches only require the existence of a checkpoint server deployed on a set of strongly safe resources. This server stores the dataflow graph of the execution provided by the *Kernel for Adaptive, Asynchronous Parallel and Interactive* – KAAPI – application programming interface. KAAPI is a C++ library that allows to program and execute multi-threaded computations with dataflow synchronization between threads. In addition, $EMCT$ requires the deployment of verifiers that could securely re-execute some tasks in a trusted way, therefore on strongly secure resources.

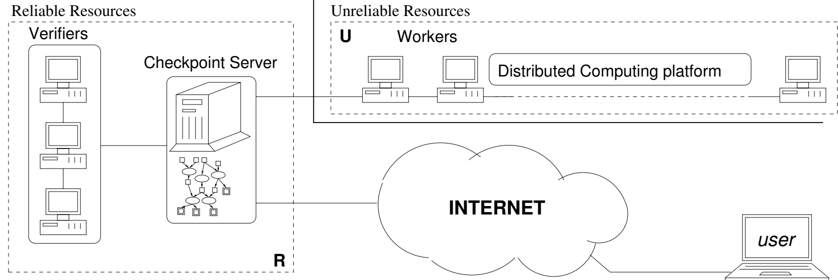


Fig. 2. Overview of the SAFESCALE computing platform

The elements presented in this section finally lead to the SAFESCALE computing platform presented in Figure 2 in which the resources have been divided in two classes: a limited number of strongly safe resources that host the checkpoint server and the verifiers and the other resources, mentioned as "unsafe", which constitute the real computing grid and which are scattered among the different institutions (for example, the different hospitals involved in the experiment described in section 4). It remains to detail the effective construction of the safe resources used in the proposed architecture. This is one of the contributions of this article and the purpose of the next section.

3 Building Strongly Secured Resources

Building safe resources is one of the most important challenge of system administrators. The solution used generally combines various software solutions to make the system more robust.

Software Components. We first assume that the software elements mentioned in the previous section are deployed. Moreover even if the software components are secure, there is an asymmetry in the trust for all the previous techniques: the aim is to protect the computing infrastructure from the program execution and there is no way for the users to have a certified and protected execution.

There is still a big issue with a full software approach: the operating system on the computing nodes have the full control of the hardware and the software running on the nodes. It is very useful to build very complex and powerful computing environment but it may be very dangerous too for the (foreign) users of these computing nodes. A computing node can easily discard a foreign process with all its data if it looks malicious or exploits too many resources. In addition, even if the operating system environment would be bug free, a foreign process needs to be completely confident in the local software environment: this one can discard the process, the data, read the program and the data, modify both, execute the program step-by-step, and so on. Although DoS are unavoidable (the local administrator could decide to switch off the power supply of the computer anyway), computers should have a mechanism to mitigate this excess of power

used in a malicious way or to signal a running process that a kind of DoS occurred (such as a system call hijacking).

In the real life, operating systems and distributed computing environment are huge pieces of software and it is unavoidable to have many bugs in them. Thus, adding pieces of hardware to protect some processes from other parts running out of their rails is quite interesting.

Hardware Components. During the last few years, several hardware architectures such as XOM [17], AEGIS [18] and CRYPTOPAGE [19] have been proposed to provide computer applications with a secure computing environment. These architectures use memory encryption and memory integrity checking to guarantee that an attacker cannot disturb the operation of a secure process, or can only obtain as little information as possible about the code or the data manipulated by this process even with some external physical attacks.

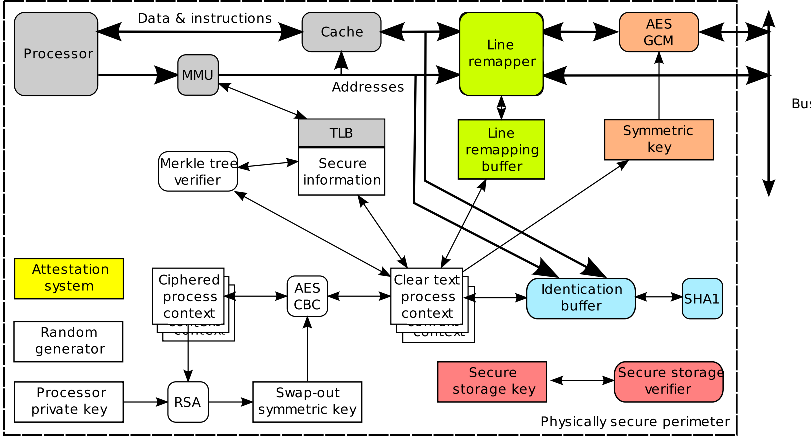


Fig. 3. Simplified CRYPTOPAGE secure high-performance processor

Some secrets can also leak out through the address bus of the processor (an attacker can monitor the control flow graph of a running program and infer algorithms or ciphering keys for example), some approaches try to cipher the address bus more [20] or less [21]. Recently, we have proposed a processor architecture that combines opaque secure mode execution with efficient memory encryption and verification, resistant to replay-attack, with dynamic random remapping of cache lines in memory page to hide memory usage and to avoid address tracing [22]. The simplified architecture is represented on Figure 3. The white boxes are the ones we have added to a plain processor architecture (the gray boxes). Rounded boxes are computing or function elements and the squared boxes are some storage elements.

A process cannot be stolen or tapped since its execution context is enciphered with the public key of a given target processor. The confidentiality of the code and the data is guaranty by a cipher (using a random session key) between the

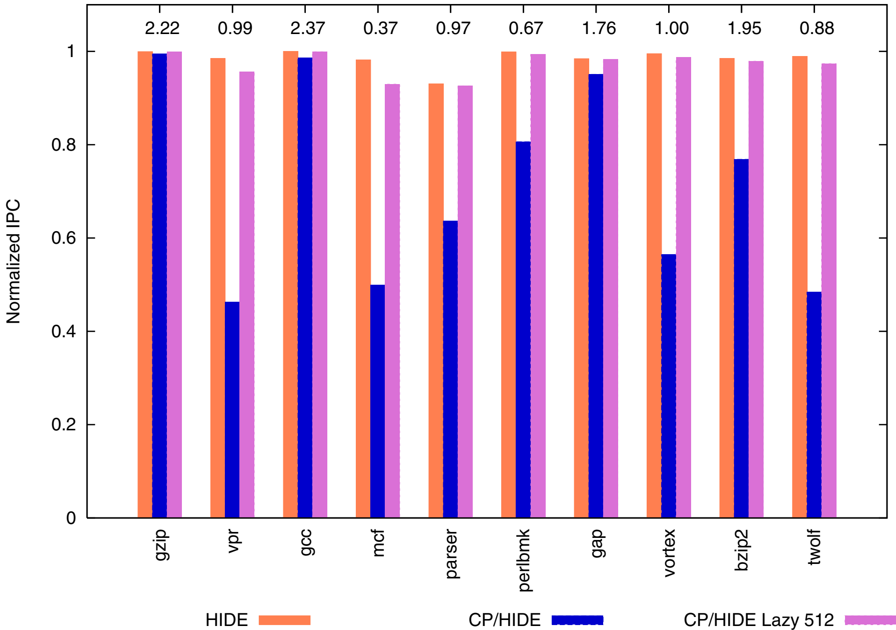


Fig. 4. Relative efficiency of different secure processor implementations (HIDE and two CryptoPage versions) relative to a plain processor (normalized to 1)

internal cache and the memory. A data Hide-like address remapper [20] is added to shuffle data each time a memory page is read into the processor. At this page level, an efficient mechanism is also added to avoid a replay attack (when an attacker replay an old data written and authenticated by the processor).

We implemented this architecture in the SimpleScalar simulator to have some quantitative results. The performance on the SPEC CPUInt2000 benchmark are represented on Figure 4 to compare HIDE [20] (which provide only address ciphering) and CryptoPage [22] (providing both address and data ciphering), the latest being evaluated without or with lazy verification through speculative insecure execution. This benchmark shows only an average slow-down of around 3% for Cryptopage with address and data ciphering and lazy verification through speculative insecure execution. This last mode is the one used on the SAFESCALE platform for the strongly secured resources. It is finally important to note that in our proposal, a foreign process needs to ultimately trust the manufacturer of the processor which could have some wire-tapping or key-escrow features in it, or more probably hardware bugs too.

4 Validation on a Healthcare Application

To validate the presented infrastructure, the following security-demanding healthcare application is considered:

- a picture A is compared to a set $\mathcal{S}$ of pictures stored in a distributed database;
- based on meta-data information, some images $X \in \mathcal{S}$ are extracted and compared to A; the result of this comparison is a score $s_A(X)$ that measures the correlation between pictures A and X;
- finally, the sorted results are brought to the end users.

Among concrete instances of such a generic application is the RAGTIME software that uses medical image comparison within PACS (Picture Archiving and Communication Systems) to detect rare and hard to predict diseases [2]. Due to numerical uncertainties brought by score computations, relevant results are obtained if several images in $\mathcal{S}$ are identified as matching picture A. Also, such an application may directly takes benefit from the computational power of a global computing infrastructure:

- the number of pairwise comparison scores to compute is huge and scores may easily be computed in parallel;
- for a given user that submits a picture A interactively to the system, the system usage is irregular, from high when it submits a picture to zero when the user has no picture to score. Thus, federating resources from several users in a single grid, positively contributes to increase the system throughput for any users;
- the application tolerates few errors, which are expensive to prevent in a grid context. Indeed, the major result is to find several images in $\mathcal{S}$ that are correlated to A, and thus bring together information on A. In this context, the fact that only few scores have been faked (let say one or two) does not affect the result. The critical point is here to ensure that almost all computations have been correctly performed.

Experimental setup. This application has been run on the SAFESCALE computing platform presented in section 2, deployed on Grid5000 (an experimental grid platform gathering nine sites in France). This simulates up to $n_p = 160$ "unsafe" computing resources that execute in parallel n comparison tasks, each resulting in a similarity score. A single strongly secured resource has been used for this experiment, with (resp. without) the hardware components described in section 3. Each configuration will be further referred as *Config. 1* (resp. *Config. 2*), the first being in fact simulated in the SimpleScalar simulator, since we do not have a real processor yet. Recall that this resource host the checkpoint server and the verifiers such that the full certification process of the computations were done on this machine. Additionally, we make this resource responsible for the sorting of the scores.

In the sequel, the error probability for the $EMCT$ certification algorithm has been set to $\epsilon = 10^{-3}$ and we try to detect the tampering of $q = 1\%$ of the score computations. Note that in this case, the certification process consists in the re-execution of $N_{\epsilon,q} = 688$ comparison tasks [1], all conducted on the single "safe" resource. The application is run in three test cases where the total number of comparison tasks is successively $n = 1000$, 10000 or 100000.

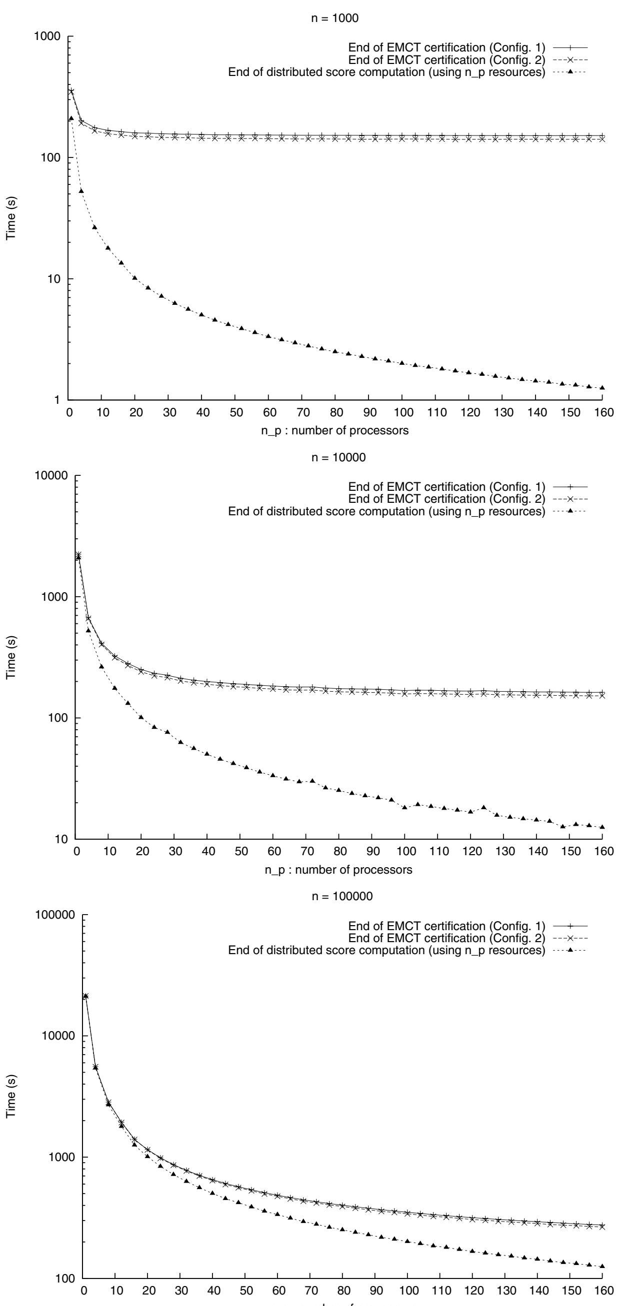


Fig. 5. Certification overhead with n comparison tasks

Distributed scores computation based on work-stealing. Since the number of similarity scores to be computed on the grid is huge, having a centralized allocation strategy (when a processor becomes idle, it contacts the master processor to obtain a new computation task) introduces contention and inefficiency. Instead, a distributed solution based on KAAPI work-stealing has been developed. When a processor becomes idle, it picks a victim at random and steals about half of is computation queue. Such a solution ensure high performances on a global architecture where processor speed may vary. Indeed, let $\Pi(t)$ be the instantaneous computation speed (i.e the number of unitary operations per time unit) and $W_A(X)$ be the number of unit operations to compute to score picture X against A; then the time T_p to carried out the whole computation is, with high probability, lesser than $\frac{\sum_{X \in \mathcal{S}} W_A(X)}{\Pi(t)} + \mathcal{O}(W_A(X) + \log n)$ which is merely optimal [23].

Experimental results. The results of our experiments are depicted in Figure 5. They illustrate the limited overhead induced by the addition of the hardware components to design the strongly secure resources. Indeed, the overhead is at most 7.4%, which corresponds to the worst case on the SpecINT 2000 benchmark. This is of course a very encouraging result.

5 Conclusion

Even if security in grid infrastructures is a major research area for a decade, the fact that resources that compose the grid cannot be fully trusted prevents a wide acceptance of such systems as a cheaper computation platform for high-valued applications. The corruption either comes from hardware issues (such as network disconnection) or from malicious acts (using malwares and software vulnerabilities) in order to alter the computation and consequently its result.

The computing platform designed in the SAFESCALE project ensures a correct and safe computation by combining efficient result-checking and fault-tolerance algorithms with a limited number of strongly secure resources. The result-checking algorithms rely on stochastic verifications by running again on trusted verifiers, some parts of the global computation chosen by studying the data-flow graph of the application. Since verifications can also be tampered, these tasks need to be run on different computers in different entities to have forgeries likely detected. But by using a limited number of strongly secure hardware resources, the sensible and time-consuming tasks of verification can be executed on some remote tamper-proof nodes with no fear about any attacker changing these results. According to the number of trusted resources available in the grid, our method needs more or less redundant verifications.

Apart from providing generic guidelines for the definition of a secure computing grid, this paper focus on the effective construction of the strongly secured resources required by the SAFSCALE computing platform. Our proposal depends on both software and hardware mechanisms which is relatively novel. This architecture has been validated on a medical application consisting in similarity computations between medical images. Our experiments demonstrate the

low overhead induced by the hardware components proposed for the definition of strongly secured resources.

Perspectives of the work presented in this article include concrete hardware implementations to validate the simulations conducted in our experiments. Additionally, we are now working on the automatic parallelization of application into the KAAPI model, based on the PIPS source-to-source compiler [24].

Acknowledgments

We would like to thanks all the members of the SAFESCALE project. This work is supported by the ANR (French National Research Agency) project SAFESCALE-BGPR ANR-05-SSIA-005 and the French Department of Defense.

References

1. Krings, A., Roch, J.L., Jafar, S., Varrette, S.: A Probabilistic Approach for Task and Result Certification of Large-scale Distributed Applications in Hostile Environments. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 323–333. Springer, Heidelberg (2005)
2. Varrette, S., Roch, J.L., Montagnat, J., Seitz, L., Pierson, J.M., Leprévost, F.: Safe Distributed Architecture for Image-based Computer Assisted Diagnosis. In: IEEE 1st International Workshop on Health Pervasive Systems (HPS 2006), Lyon, France (June 2006)
3. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. International J. of Supercomputer Applications and High Performance Computing 11(2), 115–128 (Summer 1997)
4. Molnar, D.: The SETI@Home Problem (November 2000), http://www.acm.org/crossroads/columns/onpatrol/september2000.html
5. Necula, G.C., Lee, P.: Proof-carrying code. Technical Report CMU-CS-96-165, School of Computer Science, Pittsburg (1996)
6. Aucsmith, D.: Tamper resistant software: An implementation. In: Information Hiding, pp. 317–333 (1996)
7. Sarmenta, L.F.G.: Protecting Programs from Hostile Environments: Encrypted Computation, Obfuscation, and Other Techniques. In: Area exam paper, Dept. of Electrical Engineering and Computer Science. MIT, Cambridge (1999)
8. Collberg, C.S., Thomborson, C.: Watermarking, tamper-proofing, and obfuscation - tools for software protection. In: IEEE Transactions on Software Engineering, vol. 28, pp. 735–746 (August 2002)
9. Loureiro, S., Bussard, L., Roudier, Y.: Extending tamper-proof hardware security to untrusted execution environments. In: CARDIS, pp. 111–124 (2002)
10. Varrette, S., Georget, S., Montagnat, J., Roch, J.-L., Leprevost, F.: Distributed Authentication in GRID5000. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 314–326. Springer, Heidelberg (2005)
11. Dagorn, N., Bernard, N., Varrette, S.: Practical Authentication in Distributed Environments. In: IEEE (ed.) IEEE International Computer Systems and Information Technology Conference (ICSIT 2005), Sheraton Hotel, Alger, July 19–21 (2005); Still waiting for precisions on proceedings

12. Neuman, C., Ts'o, T.: Kerberos: An authentication service for computer networks. IEEE Communications Magazine 32(9), 33–38 (1994), http://gost.isi.edu/publications/kerberos-neuman-tso.html
13. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A Security Architecture for Computational Grids. In: Fifth ACM Conference on Computer and Communications Security Conference, San Francisco, California, November 3-5, pp. 83–92 (1998)
14. Jafar, S., Varrette, S., Roch, J.L.: Using Data-Flow Analysis for Resilence and Result Checking in Peer to Peer Computations. In: IEEE (ed.): IEEE DEXA 2004 - Workshop GLOBE 2004: Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems, Zaragoza, Spain, pp. 512–516 (September 2004)
15. Varrette, S.: Sécurité des Architectures de Calcul Distribué: Authentification et Certification de Résultats. Ph.D thesis, INP Grenoble et Université du Luxembourg (September 2007); Version beta en cours de review
16. Roch, J.L., Varrette, S.: Probabilistic Certification of Divide & Conquer Algorithms on Global Computing Platforms. Application to Fault-Tolerant Exact Matrix-Vector Product. In: Proceedings of the ACM International Workshop on Parallel Symbolic Computation 2007 (PASCO 2007), Ontario, Canada. ACM, New York (2007)
17. Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., Horowitz, M.: Architectural support for copy and tamper resistant software. In: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX), pp. 168–177 (October 2000)
18. Suh, G.E., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: AEGIS: Architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17th International Conference on Supercomputing (ICS 2003), pp. 160–171 (June 2003)
19. Keryell, R.: Cryptopage-1: vers la fin du piratage informatique? In: Symposium d'Architecture (SympA'6), Besançon, France, pp. 35–44 (June 2000)
20. Zhuang, X., Zhang, T., Pande, S.: HIDE: an infrastructure for efficiently protecting information leakage on the address bus. In: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI), pp. 72–84. ACM Press, New York (2004)
21. Duc, G., Keryell, R., Lauradoux, C.: CRYPTOPAGE : Support matériel pour cryptoprocessus. Technique et Science Informatiques 24, 667–701 (2005)
22. Duc, G., Keryell, R.: CRYPTOPAGE: an efficient secure architecture with memory encryption, integrity and information leakage protection. In: Proceedings of the 22th Annual Computer Security Applications Conference (ACSAC 2006). IEEE Computer Society, Los Alamitos (2006)
23. Roch, J.L., Traore, D., Bernard, J.: On-line adaptive parallel prefix computation. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 841–850. Springer, Heidelberg (2006)
24. Ancourt, C., Coelho, F., Creusillet, B., Keryell, R.: How to add a new phase in PIPS: the case of dead code elimination. In: Proceedings of the Sixth Workshop on Compilers for Parallel Computers (CPC 1996), Aachen, Germany, pp. 19–30 (December 1996)

Simbatch: An API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems

Y. Caniou[1,⋆,⋆⋆] and J.-S. Gay[2,⋆,⋆⋆,⋆⋆⋆]

[1] LIP-ÉNS de Lyon, Université Claude Bernard de Lyon
yves.caniou@ens-lyon.fr
[2] LIP-ÉNS de Lyon
Jean-Sebastien.Gay@ens-lyon.fr

Abstract. In this paper, we describe Simbatch, an API which offers core functionalities to realistically simulate parallel resources and batch reservation systems. The objective is twofold: proposing at the same time a tool to efficiently predict parallel resources usage based on their simulations, and to realistically study Grid scheduling heuristics that may be embedded in a Grid middleware or in a tool that deploys it. Indeed, such predictions can be used in a Grid middleware both for scheduling purposes, and to dynamically tune moldable applications in function of the load of the chosen parallel resource in place of the Grid user. Simbatch simulation experiments show an average error rate under 2% compared to *real life* experiments conducted with the OAR batch manager.

Keywords: Performance prediction, Batch systems simulation, Grid simulation, Scheduling.

1 Introduction

Nowadays Grids are built on a clusters hierarchy model, as used for example by the two projects EGEE [1] and GRID'5000 [9]. The production platform in the EGEE project (*Enabling Grids for E-science in Europe*) aggregates more than 100 sites spread over 31 countries. GRID'5000 is the French Grid for the research, which aims to own 5000 nodes spread over France (9 sites are actually participating).

Parallel computing resources are generally managed via a batch reservation system also called *batch scheduler*. Users wishing to submit parallel tasks to the resource have to write *scripts* which notably describes the number of required nodes and the walltime of the reservation. They are generally answered the starting time of their jobs.

The accessibility to the aggregated power of a federation of computing resources requires mechanisms to monitor, handle/submit jobs, etc. This can be done with the help of Grid middleware such as DIET [10] or NetSolve [11]. They aim to offer to Grid users

⋆ This work is supported by the LEGO project ANR-05-CIGC-11.
⋆⋆ This work is supported by the REDIMPS project JST-CNRS.
⋆⋆⋆ This work is supported by the cluster Rhône Alpes
[1] http://public.eu-egee.org/

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 223–234, 2009.
© Springer-Verlag Berlin Heidelberg 2009

the capacity to efficiently solve problems, while hiding the complexity of the platform. In order to efficiently exploit the resource, Grid middleware should map the computing tasks according to local scheduler policy and availability. There is consequently a two-level scheduling: one at the Grid middleware level and the other one at the batch level. Neither the conception and validation of such algorithms nor their implementation are obvious. First, the execution of *large scale* experiments monopolizes the resources and cannot be reproduced. So it seems to be necessary to define common bases in order to simulate them and draw their profiles before trying to realize them in real life. Second, to efficiently schedule real life experiments, a Grid middleware must be able to get performance estimations on parallel resources. These have also to be used to dynamically tune parallel jobs in accordance with the parallel resource availability [12,6].

Contributions of this work mainly focus on the conception of an API which extends the functionalities of the Grid simulator Simgrid, allowing to easily simulate parallel resources and batch system in Grid computing. Realistic models of PBS [1] (or Torque [2]) and OAR [8] are built-in. The quality of the results obtained during the validation of this work allows us to use it as a simulation-based performance prediction tool embedded in the Grid middleware DIET.

2 Background

This section briefly describes our previous work, which has led to the design and contributions presented in this paper. Grid-TLSE [13] aims to provide an International Expert System for Sparse Linear Algebra relying on an international Grid computing environment which manages French and Japanese computing resources.

The architecture of Grid-TLSE [18] has been improved to the one [7] described in Figure 1. The architecture relies on the integration of a "protocol" interoperability between the French and Japanese middleware, respectively called DIET and AEGIS. DIET, developed by the GRAAL INRIA team project at LIP / ÉNS Lyon, is built upon the client/agent/server paradigm, and provides the GridRPC standard API [21]. This Grid middleware is able to find an appropriate server (running a DIET Server Daemon, SED), according to the information given in the client's request (*e.g.,* problem to be solved, size of the data involved), the performance of the target platform (*e.g.,* server load, available memory, communication performance) and the local availability of data stored during previous computations. Scheduling, which can be application specific, is distributed over a hierarchy of agents (Master and Local Agents). AEGIS (Atomic Energy Grid InfraStructure) is the next version of the IT Based Laboratory (ITBL) [14] middleware developed by the JAEA (Japan Atomic Energy Agency). In AEGIS, supercomputers are isolated from the Internet by a firewall for security reasons. Usually, a user of the AEGIS system connects to the computers through a Web portal, which has the accessibility for all the computers within AEGIS. Therefore, the portal equips file management, job submission, or the other basic functions for computation. On the other hand, AEGIS also proposes a control API to meet the expectations of advanced users.

The new architecture of Grid-TLSE, pictured in Figure 1, involves the following mechanisms which are very similar to the standard DIET operations: (a) After the

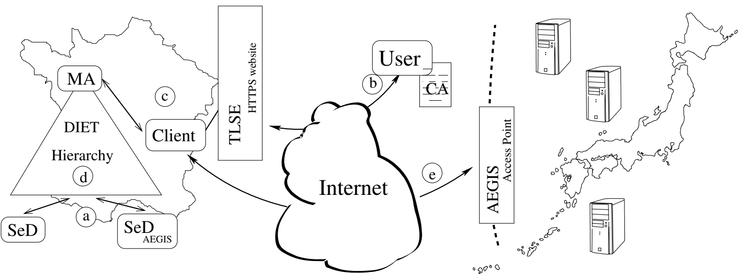


Fig. 1. Architecture of Grid-TLSE

deployment of all DIET components on Grid'5000 (including a specifically designed server daemon for the AEGIS system (SeDs$_{\text{AEGIS}}$), and composed of both binaries and configuration file(s)), each SED registers the services that it can solve to its agent in the hierarchy; (b) When a user performs a request through the secure Grid-TLSE Web portal, he must provide a certification file if he wants that his request can be executed on Japanese resources; (c) After being processed by the Grid-TLSE Web portal, the DIET client sends the corresponding request to the DIET hierarchy. The request is forwarded down (eventually pruned if the service is not available downward); (e) After having been sent back the identity of the host to contact, which can be managed either by the AEGIS middleware or by the DIET middleware, the common AEGIS-DIET client performs the call and the certificate may then be used. Once the problem solved, results are sent back to the client and so, made available to the user through the Grid-TLSE Web portal.

DIET is also able to obtain immediate information on parallel resources in order to perform cycle-stealing and online tunable parallel moldable job (with the possibility to set the number of processors to use at launch time) with as case of study, the sparse matrix solver PASTIX library [6]. Cycle-stealing policy was clearly dependent on the context of the work (analysis of a whole set of tasks without strict constraint on the experiment finishing date), and the work only used the immediate availability of the platform to tune the parallel jobs.

To improve these works, the DIET Grid middleware need estimations on when a job can be executed by a batch scheduler, which is dependent on the number of resources being requested. To be efficient, this number has to be chosen taking into account the jobs that may soon release reserved nodes which can benefit to the submitted parallel job (which would be launched later but with a smaller expected completion time). Furthermore, if a slot can be used depending on the batch scheduling policy (Conservative Backfilling for example), the Grid middleware may benefit of such information for its decisions.

Hence, Simbatch has been designed to be used as a simulation-based performance prediction tool to be used within DIET. Provided with information on the parallel system state, it takes into account the scheduler policy to instantly respond the different idle slots, with the number of processors that should be available as well as the duration of the idle slots.

3 Grid Simulators

There are numerous Grid simulators. Amongst them we can cite Bricks [24] for the simulation of client-server architectures; OptorSim [5], created for the study of scheduling algorithms dedicated to the migration and replication of data; GridSim [23] and Simgrid [19], which are by definition toolkits that provide core functionalities for the simulation of distributed applications in heterogeneous distributed environments.

Nonetheless, except for GridSim and Simgrid, systems are not generic [16,22]: they do not provide any API of reusable functions; moreover in an attempt to keep their study simple the employed scheduling policy is always *First Come First Served*; at last, they usually only implement sequential tasks, *e.g.,* they do not model *parallel tasks*.

If the GridSim toolkit covers several mandatory functionalities, it is hard to use the same code to at the same time address scheduling heuristic studies and performance predictions that can be used online to dynamically tune parallel applications according with the resource load. Furthermore, as it is written in JAVA, the use of GridSim, if feasible, is contradictory within the context of the lightweight deployment of the DIET Grid middleware.

Thus, we have chosen to integrate Simbatch [3] in the Simgrid toolkit to embed in DIET an efficient performance prediction tool, in order to improve its quality of service. In addition, its design also eases the conception and analysis of Grid scheduling heuristics.

4 The Simbatch API

Simbatch is a C API consisting of 2000 lines of code. It uses data types and functionalities provided by the Simgrid library to model clusters and batch systems. Simbatch provides a library containing already three scheduling algorithms [20]: *Round Robin* (RR), *First Come First Served* (FCFS) and *Conservative Backfilling* (CBF). The API is designed to easily let the user integrate its own algorithms. In order to visualize the algorithm behavior, a compliant output with the Pajé [4] software is available allowing the draw of the Gantt chart of the execution.

4.1 Modeling

Clusters consist of a frontal computer relied to interconnected computing resources following a specific topology. Resources of a cluster cannot be usually accessed directly from outside the cluster: communications must be done through the frontal. The batch manager system is run on the frontal node. Every jobs running on the nodes must have been submitted to the batch system. It receives requests from users, schedules them on the parallel resources and executes the corresponding task when needed. In this context, scheduling means that the batch scheduler must determine the starting time for each computing task and must allocate computing nodes for each of them. The computing tasks are generally parallel and could have both input and output data.

In Simbatch, a parallel task submitted by a client is modeled by the addition of different information to a Simgrid task data type such as the number of nodes, the

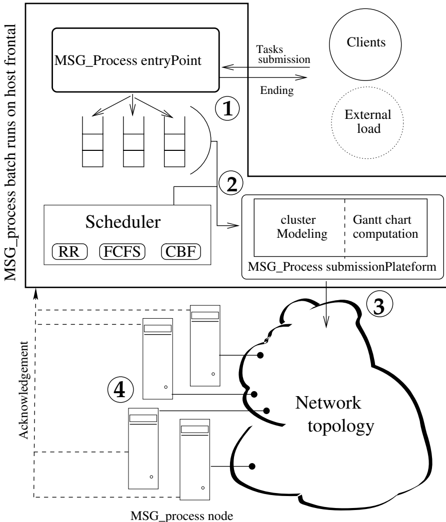


Fig. 2. Simbatch architecture

walltime, the run time. Other models are directly inherited from Simgrid. As exposed in Figure 2, the task treatment is made in the following manner:

1. The `entryPoint` process accept parallel tasks submission from the different clients and put them in the right priority queue.
2. Thanks to the modeling unit, the scheduler assigns the starting time to the tasks and reserves the computing resources needed for their execution. A global view of the cluster is obtained by calculating the Gantt chart.
3. The submission module manages the sending of each task at the starting date on the reserved resources. It controls the good respect of the reservation too. A task is killed if the walltime is exceeded.
4. Because Simbatch is build on top of Simgrid, it lets this one simulate communications and executions. When a task finishes its life cycle, an acknowledgement is sent to the `batch` process in order to update its global view of the cluster.

4.2 Using Simbatch

An experiment requires at least four files: a *platform file*, a *deployment file*, a *batch configuration file* and a file describing the tasks which will be submitted to each parallel resource, the *external load*.

Simgrid uses the *platform file* to describe resources which compose the simulated platform. It contains the description of all resources such as nodes, network links, the connectivity of the platform, etc.

Simbatch requires a *deployment file* in which the functions attached to the resources are defined. Thus, to define the use of a batch scheduler on frontal nodes, one must affect the `SB_batch` process provided by the Simbatch API to each frontal name.

Likewise, one must declare for each computing resource of each cluster the execution of the `SB_node` process provided by the Simbatch API.

The *batch configuration file* contains all information relative to each frontal node of the platform, like the number of waiting queues and the scheduling algorithm.

The *external load* is generated by the tasks submissions of the simulated Grid platform users. The file, whose name is recorded in the configuration file, describes the tasks specifications such as dates of submission, numbers of processors, walltimes, etc.

It is also possible to simulate an *internal load* for *each* batch scheduler. It aims at reproducing the submissions of tasks from clients who are directly connected to the parallel resource, *i.e.,* who are not actively participating to the simulated Grid platform. There is at most the same number of *internal load* files than the number of frontal nodes in the platform.

We give P.234 some of the files that we used for the experiments. The main code shows a client submitting a task to a batch scheduler described as follows: use of an external load, 3 priority queues, 5 nodes directly connected with a star topology and the CBF algorithm. In addition, we have also modeled the Grid'5000 node of Lyon [2].

5 Experimental Validation

Tasks generation. In order to validate the results obtained with Simbatch, we have built a workload generator using the GSL library [15]. It uses a Poisson's law with parameter $\mu = 300$ to generate inter-arrival time. Tasks specifications are determined by flat laws. Thus, CPU numbers are drawn from $U(1;7)$, execution durations from $U(600;1800)$ and walltimes are obtained by balancing the corresponding execution duration by a random number drawn from $U(1.1;3)$.

Some experiments have been conducted with communicating tasks. They are all independent but require the communication of input data from the frontal node to one node allocated to the parallel task, and the communication of the output data back to the frontal node. In order to do this, we have created 6 files with a size of respectively $1, 2, 5, 10, 15, 20$ MB . One of this file is chosen randomly by a uniform law to be transfered as input data, while another file is chosen in the same manner to be transfered as output data.

Real experiments platform. OAR [8] is a batch reservation system developed by the project `MESCAL` in Grenoble. It is deployed on each site of the Grid'5000 platform. The scheduling algorithm used is CBF.

The 1.6 version of OAR has been installed on a cluster made of 1 frontal and 7 servers SuperMicro 6013PI equipped with a XEON processor at 2.4 GHz, each of them relied to a 100 Mbits/sec switch.

Protocol of experimentation. We have modeled the real platform by creating computing resources connected in a star topology. Then, thanks to our load generator, we have submitted the same load to both platforms (real and simulated). In this purpose, we have created a MPI computing task whose duration is given as parameter. The task

[2] `http://graal.ens-lyon.fr/simbatch`

is executed between two calls for time measuring in an OAR script. The precision of the time measure is about 1 second, so it is negligible compared to the task duration.

6 Results and Discussion

6.1 Validation of the Integrated Scheduling Algorithms

We show in Figure 3 the result obtained for one simple experience, described in Figure 1. One can see the Pajé Gantt chart on the top and beneath, the one obtained with the Drawgantt OAR. The tasks execution order is strictly the same (as it has always been, tested with a extensive set of experiments [17]): task 3, 4, 5 benefit from the *backfilling* and start their execution before task 2 which needs every nodes of the cluster. However, we can point out that Simbatch doesn't necessarily allocate the same nodes than OAR (task 3).

6.2 Accuracy of Simbatch Simulations

Two sets of experiments have been conducted with the second protocol: only computing tasks are involved in the first one, as the second one uses exclusively communicating tasks.

Experiments involving computing tasks. We have run numerous experiments for the first set of experiments, representing about 130 hours of computing on the cluster. Only computing tasks are involved. We present here a representative experiment for this set.

Table 1. Data used for experiment 1

Tasks	1	2	3	4	5
Processors number	1	5	2	1	3
Submission date	0	600	1800	3600	4200
Run time	10800	3300	5400	4000	2700
Reservation time	12000	4000	7000	5000	3500

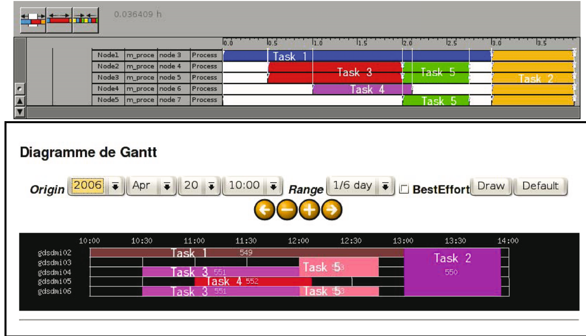


Fig. 3. Gantt chart for experiment 1: Simbatch (top), OAR (bottom)

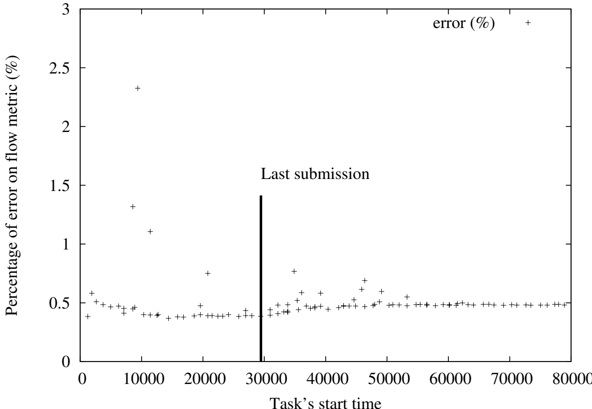


Fig. 4. Error ratios for an experiment scheduled with the Simbatch CBF and the real-life CBF implemented in OAR

The experiment consists in submitting the same set of 100 computing tasks to the real platform and to the simulated platform: results obtained with OAR was exactly 80392 seconds (about 22h) while the simulator gave us 80701 seconds. So the total execution time difference is only 308 seconds. It represents an error rate of 0.38%. This difference is mainly due to the mechanisms for interrogating the Mysql database, submitting tasks via ssh, etc., of OAR.

Figure 4 shows the error rate obtained for the flow metric in function of the tasks *execution* date. The flow of a task is the *time spent in the system*, *i.e.,* results from the addition of the waiting time passed in the queue, of the run time and of the communication costs. We can point out that the error rate is constant and generally below 1%, which is negligible compared to the precision of the measure. We can point out that for each experiment we have few tasks with an error rate above the 1%. This phenomenon is *not due to a scheduling error* due to some time precision here. In fact, some shorter and small tasks (time and processor) can enter the system and take advantage of the backfilling both with Simbatch and OAR. Because of the small gap between Simbatch and OAR starting time (thus between their ending time as well), the task begins a little later in reality, which can represent up to 15% and has only been observed once in our experiments (the second maximum observed is 6%).

An arrow is also drawn at time 29454: it represents the date of the last *submission*. In a dynamic environment, if we give to Simbatch *every specification* of a set of tasks submitted to a batch scheduler, then Simbatch should be able to make a reliable prediction on the execution of this set.

Experiments involving communicating tasks. Since we obtained excellent results for the simulation of batch scheduler for parallel tasks without communication costs, we have decided to go further and we have tested Simbatch with experiments involving communication costs. Hence, we transfer some data from the frontal node to one of the allocated nodes for the parallel task. Once the computation done, we transfer back some data from the same allocated node to the frontal.

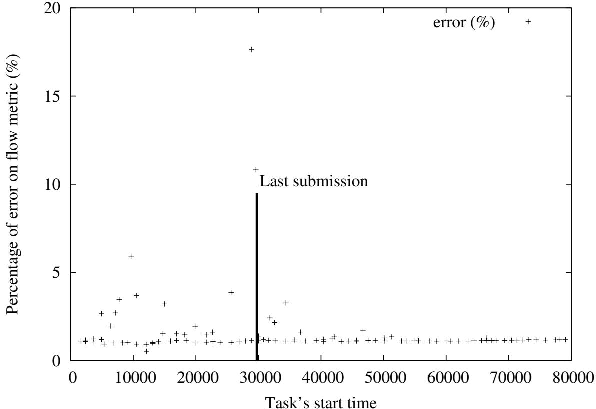


Fig. 5. Error ratios for an experiment involving communication costs scheduled with the Simbatch CBF and the real-life CBF implemented in OAR

We have run the experiments on our platform with OAR and in simulation with Simbatch. Then, we have measured the flow for each task and we have calculated the error on the flow metric between simulation and real experiments.

Figure 5 depicts one representative experiment. The error rate is low, with an average around 2%. However a few tasks have a higher error percentage. After having analysed our results, we can point out that those tasks have the same profile, *i.e.,* small computation time and few resources needed (typically 1 processor). When we analyse deeply, we can conclude that those tasks are taking advantage of the backfilling in simulation contrarily to the reality. In spite of the fact that some tasks are scheduled earlier in simulation than in reality, the impact is very small on the other task flow: the duration of the task which benefited from a backfilling represents a small percentage of other task flow.

Thus, Simbatch obtains realistic results for simulated experimental studies. It allows to easily model parallel resources managed by a batch scheduler. The good quality of its simulation shows its relevancy in the study of scheduling algorithms for the Grid. Furthermore, its use is straightforward as prediction module in a Grid middleware.

7 Conclusions and Future Works

The submission of parallel tasks by a Grid middleware is not straightforward, particularly due to the lack of profiling functionalities in batch schedulers. Nonetheless, performance estimations must be used to both efficiently schedule tasks on the Grid and tune accordingly parallel tasks with the parallel resource load and scheduling policy.

As a step in the Grid-TLSE architecture, we have designed Simbatch. It can be embedded in a Grid middleware to give accurate predictions. We have detailed those functionalities and we have specified the models we use. Then we have shown the facility for every Simgrid user to use Simbatch thanks to the examples coming from our validation work.

The main scheduling algorithms (*Round Robin*, *First Come First Served* and *Conservative BackFilling*; the last two are respectively implemented in PBS, and in MAUI and OAR) are integrated and have been validated by several simulation experiments. Moreover, we have compared results obtained from Simbatch simulations with the ones from the real-life batch scheduler OAR. Simbatch shows very good precision with an error rate in general less than 2%.

There are numerous perspectives. Among them, we want to test and eventually extend Simbatch to batch schedulers like SGE or Loadleveler; we want also to design scheduling heuristics to take advantage of such predictions and integrate them in DIET for immediate use in the Grid-TLSE system.

References

1. http://www.openpbs.org/
2. http://old.clusterresources.com/products/torque/
3. http://simgrid.gforge.inria.fr/doc/contrib.html
4. http://www-id.imag.fr/Logiciels/paje/
5. Bell, W., Cameron, D., Capozza, L., Millar, P., Stockinger, K., Zini, F.: Optorsim - a grid simulator for studying dynamic data replication strategies. Journal of High Performance Computing Applications 17 (2003)
6. Caniou, Y., Gay, J.-S., Ramet, P.: Tunable parallel experiments in a gridrpc framework: application to linear solvers. In: VECPAR 2008 International Meeting on High Performance Computing for Computational Science (2008) (to appear)
7. Caniou, Y., Kushida, N., Teshima, N.: Implementing interoperability between the AEGIS and DIET GridRPC middleware to build an International Sparse Linear Algebra Expert System. In: Second International Conference on Advanced Engineering Computing and Applications in Sciences, ADVCOMP 2008 (2008)
8. Capit, N., Da Costa, G., Georgiou, Y., Huard, G., Martin, C., Mounier, G., Neyron, P., Richard, O.: A batch scheduler with high level components. In: Cluster computing and Grid 2005, CCGrid 2005 (2005)
9. Cappello, F., Desprez, F., Dayde, M., Jeannot, E., Jegou, Y., Lanteri, S., Melab, N., Namyst, R., Primet, P., Richard, O., Caron, E., Leduc, J., Mornet, G.: Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. In: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, Grid 2005, Seattle, Washington, USA (November 2005)
10. Caron, E., Desprez, F., Fleury, E., Lombard, F., Nicod, J.-M., Quinson, M., Suter, F.: Une approche hiérarchique des serveurs de calculs. In: Calcul réparti à grande échelle, Hermés Science, Paris (2002)
11. Casanova, H., Dongarra, J.: Netsolve: A network server for solving computational science problems. In: Proceedings of Super-Computing, Pittsburg (1996)
12. Cirne, W., Berman, F.: Using moldability to improve the performance of supercomputer jobs. J. Parallel Distrib. Comput. 62(10), 1571–1601 (2002)
13. Daydé, M., Desprez, F., Hurault, A., Pantel, M.: On deploying scientific software within the Grid-TLSE project. Computing Letters 1(3), 85–92 (2005)
14. Fukuda, M.: ITBL – toward constructing a new R & D environment, vol. 55, pp. 19–23 (2002)
15. Galassi, M., Theiler, J.: The Gnu Standard Library (1996)
16. Garonne, V.: DIRAC - Distributed Infrastructure with Remote Agent Control. Ph.D thesis, Université de Méditéranée, Décember (2005)

17. Gay, J.-S., Caniou, Y.: Simbatch: an api for simulating and predicting the performance of parallel resources and batch systems. Technical Report RR2006-32, LIP ENS-Lyon, Université Claude Bernard Lyon 1, Lyon, (October 2006)
18. Kushida, N., Suzuki, Y., Teshima, N., Nakajima, N., Caniou, Y., Daydé, M., Ramet, P.: Toward an International Sparse Linear Algebra Expert System by interconnecting the ITBL computational Grid with the Grid-TLSE platform. In: VECPAR 2008 International Meeting on High Performance Computing for Computational Science (2008) (to appear)
19. Legrand, A., Marchal, L., Casanova, H.: Scheduling distributed applications: the simgrid simulation framework. In: IEEE Computer Society (ed.) 3rd International Symposium on Cluster Computing and the Grid, p. 138. IEEE Computer Society, Los Alamitos (2003)
20. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. In: IEEE (ed.) IEEE Trans. Parallel and Distributed Systeme, pp. 529–543. IEEE, Los Alamitos (2001)
21. Nakada, H., Matsuoka, S., Seymour, K., Dongarra, J., Lee, C., Casanova, H.: A GridRPC Model and API for End-User Applications (December 2003)
22. Ranganathan, K., Foster, I.: Decoupling computation and data scheduling in distributed data-intensive applications. In: 11th IEEE International Symposium on High Performance Distributed Computing, HPDC-11 (2002)
23. Sulistio, A., Cibej, U., Venugopal, S., Robic, B., Buyya, R.: A toolkit for modelling and simulating data grids: An extension to gridsim (accepted December 3, 2007) (in press)
24. Takefusa, A., Casanova, H., Matsuhoka, S., Berman, F.: A study of deadline for client-server sytems on the computational grid. In: 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), pp. 406–415 (2001)

Annexe A

```
<?xml version='1.0'?>
<!DOCTYPE platform_description SYSTEM "surfxml.dtd">
<platform_description>
  <process host="Client" function"client">
     <argument value="0" />
     <argument value="0" />
     <argument value="0" />
     <argument value="Frontale" /> <!-- Connection -->
  </process>
  <!-- The Scheduler process (with some arguments) -->
  <process host="Frontale" function="SB_batch">
     <argument value="0" />       <!-- Number of tasks -->
     <argument value="0" />       <!-- Size of tasks -->
     <argument value="0" />       <!-- Size of I/O-->
     <argument value="Node1" />  <!-- Connections-->
     <argument value="Node2" />
     <argument value="Node3" />
     <argument value="Node4" />
     <argument value="Node5" />
  </process>
  <process host="Node1" function="SB_node"/>
  <process host="Node2" function="SB_node"/>
  <process host="Node3" function="SB_node"/>
  <process host="Node4" function="SB_node"/>
  <process host="Node5" function="SB_node"/>
</platform_description>
```

Deployment file

```
<?xml version='1.0'?>
<!DOCTYPE platform_description SYSTEM "surfxml.dtd">
<platform_description>
  <cpu name="Client" power="97.34000000000000"/>
  <!-- One scheduler for one cluster of five nodes -->
  <!-- Power of the batch is not important         -->
  <cpu name="Frontale" power="98.094999999999999"/>
  <cpu name="Node1" power="76.296000000000006"/>
  <cpu name="Node2" power="76.296000000000006"/>
  <cpu name="Node3" power="76.296000000000006"/>
  <cpu name="Node4" power="76.296000000000006"/>
  <cpu name="Node5" power="76.296000000000006"/>
  <!-- No discrimination for the moment-->
  <network_link name="0" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="1" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="2" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="3" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="4" bandwidth="41.279125" latency="5.9904e-05"/>
  <network_link name="5" bandwidth="41.279125" latency="5.9904e-05"/>
  <!-- Simple topologie -->
  <route src="Client" dst="Frontale"><route_element name="0"/></route>
  <route src="Frontale" dst="Node1"><route_element name="1"/></route>
  <route src="Frontale" dst="Node2"><route_element name="2"/></route>
  <route src="Frontale" dst="Node3"><route_element name="3"/></route>
  <route src="Frontale" dst="Node4"><route_element name="4"/></route>
  <route src="Frontale" dst="Node5"><route_element name="5"/></route>
  <!-- Bi-directionnal-->
  <route src="Node1" dst="Frontale"><route_element name="1"/></route>
  <route src="Node2" dst="Frontale"><route_element name="2"/></route>
  <route src="Node3" dst="Frontale"><route_element name="3"/></route>
  <route src="Node4" dst="Frontale"><route_element name="4"/></route>
  <route src="Node5" dst="Frontale"><route_element name="5"/></route>
</platform_description>
```

Platform description file

```
<?xml version="1.0" ?>

<config>
        <!-- Global settings for the simulation -->
        <global>
                <file type="platform">platform.xml</file>
                <file type="deployment">deployment.xml</file>
                <!-- Paje output : suffix has to be .trace -->
                <file type="trace">simbatch.trace</file>
        </global>

        <!-- Each batch deployed should have its own config -->
        <batch host="Frontale">
                <plugin>librrobin.so</plugin>
                <!-- Internal Load-->
                <wld>./workload/seed/1.wld</wld>

                <priority_queue>
                        <number>3</number>
                </priority_queue>
        </batch>

        <!-- Other batchs -->
</config>
```

Configuration of the simulated batch system

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <msg/msg.h>
#include <simbatch.h>
#define NB_CHANNEL 10000

/* How to create and send a task */
int client(int argc, char * argv)
{
    job_t job=calloc(job_t, sizeof(job));
    m_task_t task=NULL;

    strcpy(job->name, "tache");
    job->nb_procs = 3; job->priority = 1;
    job->wall_time = 600; job->requested_time = 1800;
    job->input_size = 100; job->output_size = 600;
    task = MSG_task_create(job->name, 0, 0, job);
    MSG_task_put(task ,MSG_get_host_by_name("Frontale")
                 ,CLIENT_PORT);
}

int main(int argc, char ** argv)
{
   SB_global_init(&argc, argv);
   MSG_global_init(&argc, argv);

   /* Open the channels */
   MSG_set_channel_number(NB_CHANNEL);
   MSG_paje_output("simbatch.trace");

   /* The client who submits requests (write your own)
    * Params have to be called with the same name */
   MSG_function_register("client", client);

   /* Register simbatch functions */
   MSG_function_register("SB_batch", SB_batch);
   MSG_function_register("SB_node", SB_node);

   MSG_create_environment("platform.xml");
   MSG_launch_application("deployment.xml");

   MSG_main();

   /* Clean everything up */
   SB_clean();
   MSG_clean();

   return EXIT_SUCCESS;
}
```

Simgrid main code

Analysis of Peer-to-Peer Protocols Performance for Establishing a Decentralized Desktop Grid Middleware

Heithem Abbes[1,2] and Jean-Christophe Dubacq[1]

[1] LIPN — UMR CNRS 7030 — Institut Galilée — Université Paris-Nord
99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France
Tel.: +33(0)1 49 40 35 78, Fax: +33(0)1 48 26 07 12

[2] École Supérieure des Sciences et Techniques de Tunis, Unité de recherche UTIC
5, Av. Taha Hussein, B.P. 56, Bab Mnara, Tunis, Tunisia
Tel.: (+216) 71 496 066, Fax: (+216) 71 391 166
{heithem.abbes,jean-christophe}@lipn.univ-paris13.fr

Abstract. The Desktop Grid technology consists mainly in exploiting personal computer, geographically dispersed, to deliver massive compute power to investigate complex and demanding problems in a variety of different scientific fields. However, as resources number increases, the need for scalability and decentralization becomes more and more essential. Since such properties are exhibited by Peer-to-Peer systems, we aim at using them to create a decentralized desktop grid middleware. Nevertheless, in order to judge the efficiency of such P2P library, an experimental performance evaluation of the provided functionalities is unavoidable. Very few analysis of this kind have been reported, as most evaluations are limited to complexity analysis and to simulations. Such experimental analysis are important, especially when using P2P tools in grid computing context, when applications may have precise efficiency requirement. In this paper, we focus on three libraries: Bonjour, Avahi and Pastry, which provide generic API intended to serve as basis for specialized P2P applications. We perform a performance evaluation of the scalability and their capacity to register and browse an important number of services over 300 hosts in Grid'5000 for recent versions of Pastry, Avahi and Bonjour. We provide detailed analysis explaining the behavior of each library related to two criteria: the elapsed time for registration services and the needed time to discover services. Our aim is to choose the most adequate protocol for creating a decentralized middleware for desktop grid.

1 Introduction

Grid Computing aim at providing a powerful infrastructure based on the aggregation of large numbers of resources spanning multiple organizations. This concept is rapidly emerging as the dominant paradigm for distributed problem solving for a wide range of application domains. The middleware of grid computing is, however, so complex that it needs a lot of effort to maintain. Therefore

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 235–246, 2009.
© Springer-Verlag Berlin Heidelberg 2009

it is natural, that single persons do not offer their resources but all resources are maintained by institutions, where professional system administrators take care of the environment and ensure the availability of the Grid. Examples of such Grid infrastructures are the TeraGrid [1], EGEE [2] and Open Science Grid (OSG) [3]. Complementary to grid computing, Desktop Grids leverage Internet connected computers to support large computations. In this kind of system, a single person can more easily add his personal computer (PC) to the grid. Desktop grid have been successfully used to address large applications with significant computational requirements, including global climate predication (Climatprediction.net), protein structure prediction (Predictor@Home), search for extraterrestrial intelligence (SETI@Home), gravitational wave detection (Einstein@Home), and cosmic rays study (XtremWeb). While the successes of the above applications do demonstrate the potential of desktop grid, existing systems are often centralized and suffer from being not able to scale due to centralized control. To bypass this, we would profit from existing decentralized P2P systems in order to organize the management of the desktop grid. Nonetheless, we face the following issue: (i)which kind of systems to choose? (ii) more exactly, what is the maximum number of PC that can be managed by a system? (iii) If one machine join the desktop grid, how many time needed to register this machine? By evaluating three "famous" P2P systems with real experimentations on the testbed GRID'5000[4], we provide quantitative answers to these questions.

Moreover, in this work, we assume that we have a high level middleware able to virtualize the network (we have no more problems with firewall and NAT) and we are able to run Bonjour, Avahi and Pastry on top of such middleware. Instant Grid / Private Virtual Cluster [5,6] is one of the candidate for network virtualization. Its main requirements are: 1) simple network configuration 2) no degradation of resource security 3) no need to re-implement existing distributed applications. Under these assumptions, it is reasonable to check if Bonjour [7,8], Avahi [9] and Pastry [10,11] can scale up. The core idea behind these protocols is to build self-organized overlay networks when nodes join a virtual organization.

The remainder of this paper is organized as follows. Section 2 motivates the choice of the three systems Bonjour, Avahi and Pastry and presents an overview of them. Section 3 describes our experimental setup to evaluate the performance of the three systems. Section 4 and section 5 provide results from our experiments in Grid'5000. Section 6 discusses related works. Section 7 summarizes our contributions and gives future works.

2 ZeroConf vs. DHT

At the moment, the trend would be to use DHT, because of the scalability property. In fact, the originality of our work is to evaluate DHT versus publish/subscribe system. We use the Pastry library to implement a DHT concept and the two implementations of ZeroConf protocol for publish/subscribe systems. The choice of Avahi and Bonjour is justified by the fact that they are two implementations of the ZeroConf protocol (Zero Configuration Networking)

which already proved its reliability in the field of the local area networks and which can be extensible on wide area networks (by using the Unicast sending protocol coupled with the DNS protocol). Among the protocols based on a DHT (Distributed Hash Table) such as CAN [12] and CHORD [13], we chose Pastry because, on the one hand, it offers the possibilities of replications and, on the other hand, because there exists an open-source implementation [11].

2.1 Bonjour

Bonjour is an implementation by Apple of the ZeroConf protocol. The goal is to obtain a functional IP network without dependency of an infrastructure comprising DHCP or DNS servers. Bonjour is structured around three functionalities: it allows the dynamic allocation of IP addresses without DHCP, it ensures the resolution of names and IP addresses without DNS and carries out the research of the services without directory server. At a technical level, Bonjour uses Link-Local addresses. When DHCP fails or is not available, link-local addressing lets a computer make up an IP address (of IPv4 type) for itself. In IPv4, link-local address is selected by means of a pseudo-random generator in a defined range of addresses (169.254.1.0-255). The checking of address uniqueness is done using three requests which are diffused on the link-local. If IP address is already used (or requested) by another machine, then the machine tries another address provided by the generator. When the machine finds a free address, it diffuses in broadcast two ARP advertisements with the source IP address containing the selected one. In fact, if at any time the machine obtains an address by DHCP then it uses this address and leaves the process of self-configuration on the link-local. Like link-local addressing, when DNS servers are unavailable or unreachable, the machines can still refer the ones with the others by name by using the protocol mDNS (multicast DNS). Bonjour uses DNS-SD protocol (DNS Service Discovery) to discover the services published in a local area network. Since DNS-SD is built on top of DNS, it works not only with mDNS but also with traditional DNS for discovering remote services.

2.2 Avahi

Avahi is a system which facilitates service discovery on a local network. It allows the programs to publish and discover services and hosts running on local network without any specific configuration. Avahi is an implementation of DNS Service Discovery and mDNS specifications for ZeroConf protocol. Avahi is mainly based on mDNS implementation for Linux. It uses D-Bus (an asynchronous library for communication between processes) for communication between user applications and system.

2.3 Pastry

In this section we briefly sketch Pastry [10]. A Pastry system is a self-organizing overlay network of nodes, where each node routes client requests and interacts

with local instances of one or more applications. Any computer that is connected to the Internet and runs the Pastry node software can act as a Pastry node, subject only to application-specific security policies. Each node in the Pastry P2P overlay network is assigned a 128-bit node identifier (nodeId). The nodeId is used to indicate a node's position in a circular nodeId space, which ranges from 0 to $2^{128} - 1$. The nodeId is assigned randomly when a node joins the system. It is assumed that nodeIds are generated such that the resulting set of nodeIds is uniformly distributed in the 128-bit nodeId space. For instance, nodeIds could be generated by computing a cryptographic hash of the node's IP address. As a result of this random assignment of nodeIds, with high probability, nodes with adjacent nodeIds are diverse in geography. Assuming a network consisting of N nodes, Pastry can route to the numerically closest node to a given key in less than $\log_2^b N$ steps under normal operation (b is a configuration parameter with typical value 4). Despite concurrent node failures, eventual delivery is guaranteed unless nodes with adjacent nodeIds fail simultaneously (is a configuration parameter with a typical value of 16 or 32).

For the purpose of routing, nodeIds and keys are thought of as a sequence of digits with base 2b. Pastry routes messages to the node whose nodeId is numerically closest to the given key. This is accomplished as follows. In each routing step, a node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the present node's id. If no such node is known, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the present node's id.

Free-Pastry is an open-source implementation of Pastry intended for deployment in the internet. We have used in our experimental tests the release 2.0.

3 Description of the Experimental Setup

The experimental platform is Grid'5000, a highly reconfigurable and controllable grid system, which gathers 9 sites geographically distributed in France. All sites are connected by RENATER network (10 Gb/s). Our tests are applied only in Orsay site, where nodes are connected by a network of 1 Gb/s. We almost used the totality of the available machines in this site (more than 300 machines). All machines have AMD Opterons processors and networks cards of 1 Gb/s.

We represent the nodes by services to build a virtual network on Grid'5000 platform. Indeed, on each machine we register a service, thus, if the service is running then the machine is connected on the network, if not (we deactivate or we remove the service) the machine is disconnected.

Our objective is to study the scalability and the response time of the tools described above. In fact, we look for the maximum number of nodes which can be supported by these tools and the response time necessary to discover a new node which has been just connected on the network (depending on the grid state). The same benchmarks criteria are applied for the three systems.

3.1 Specific Environment on Grid'5000

Grid'5000 offers an infrastructure with standard images. To run our experimental tests, we personalized an image to support Avahi, Bonjour and Free-Pastry. Thus, we created a specific Linux kernel containing the necessary packages to execute our codes. Thereafter, by using OAR [14]and Kadeploy [15] tools, we reserve and we deploy the specific image on all reserved machines according to the traditional procedure for the Grid'5000 users.

3.2 Sequential Registrations

In this test, the first step is to reserve N nodes on Grid'5000. The number N represents the maximum nodes that can be used for the experiment. Each node requests a registration for a given service at a given time. Initially, all nodes have the needed codes to request a service but are inactive. Let δ be the activation time. We activate sequentially all the requests (and we receive back an acknowledgement). Indeed, the kth request will be activated at time $k \times \delta$. We increase δ to analyze the behavior of the system when the delay between events becomes larger. Obviously, at the beginning of the test, the registration number is small, thus the registration time will be fast. We increase N until the saturation value (i.e. the registration service no longer responds for a new registration). We aim at analyzing the scalability of the system without overloading the network: in this test, only one multicast appears at a given time.

3.3 Simultaneous Registrations

In the first test, the registrations are done sequentially. This leads to a limited number of communications to exchange information. In this experiment, we stress the scalability of the system and its capacity to manage communications between registered nodes. Therefore, we request N (the number of reserved nodes) simultaneous registrations and we compute the time to complete the registration step. If we obtain a "reasonable" response time, we increase N until the saturation value. In others words, we are looking for the maximum registered nodes that the system handles when the network is overloaded by several multicast packets at the same time.

3.4 Browsing Services

The other important metric is the time needed to browse a given service. Indeed, in previous tests, we compute the registration time. We need also to compute the discovering time which is the elapsed time between the registration end of a unique service and the date at which a browser node has discovered this service. The browsing program listens any new event, i.e. a new registration or deleting services. With the two setup mentioned before, we can analyze the performance of service discovery.

4 Performance of Registration Services

4.1 Registration of Bonjour Services

Bonjour starts by making a DHCP request to obtain an address. In case of absence of DHCP, it performs one (up to 3) ARP request(s); however when the kernel is booted, an IP is already attributed, so we can not take into account the elapsed time in this first phase. Bonjour should notify all machines that the node has the service by updating all ARP caches. There could be an additional management cost related to replacements in the cache but the default size of ARP cache is 1040 entries which is higher than the number of services used in the worst case of our experiments. Consequently, we can not reasonably charge the management of ARP caches in the degradation of performances. Bonjour proceeds by checking the uniqueness of service. An initial ARP advertisement is made with one second waiting (Probe wait = 1s). If there has been not reply in the second (it is well the case, because all the services were selected with a single name), then the name is considered as unique.

In figure 1, the y-axis shows the percentage of services correctly registered at time x. The sequential registration shows a better times than simultaneous one. Indeed, the measurements taken on 308 machines give registrations times ranging between 1015 and 1030 ms. Whereas, in the simultaneous version, elapsed time varies between 1017 ms and 2307 ms. Comparing with the sequential case, we find again the same constant (1015 ms) for simultaneous registration in the first registration, but it becomes longer to reach the bus in turn and that costs about 10 to 30 ms, which generates an increase cumulated in the registration time from 1017 ms to 2307 ms.

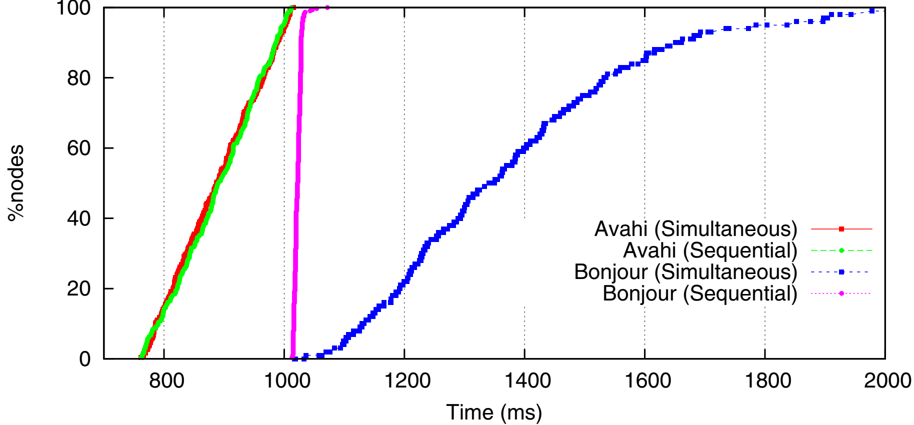


Fig. 1. Elapsed time for registrations of Bonjour and Avahi services

4.2 Registration of Avahi Services

On each node of the Avahi network, there is a running avahi-daemon. This daemon implements the two protocols mDNS/DNS-SD of Zeroconf. To register one service, Avahi publishes the service by a multicast send to all nodes, using D-Bus as transport protocol, and updates the cache of each node. These protocols showed a great effectiveness in services registration. Indeed, figure 1 shows that Avahi gives almost same elapsed times in sequential and in simultaneous tests. Elapsed time varies between 760 and 1110 ms which are better than ones given by Bonjour.

4.3 Registration of Pastry Services

Contrarily to Avahi and Bonjour, Pastry shows a great difference between sequential tests and simultaneous ones. Indeed, figure 2 shows that in simultaneous registration, until 160$^{\text{th}}$ service, elapsed time varies between 600 and 1000 ms. Beyond that, registration time increases from one registration to another to reach 320 000 ms. In addition, figure 2 shows that the sequential registration gives better times. We mention that Pastry gives reduced times in comparison to Avahi and Bonjour (30% of services, among 307, are registered in the interval 450-550 ms). Indeed, when we register a service, we require connection to the same bootstrap machine (to avoid creating several rings). Thus, in simultaneous case, the bootstrap can not answer all simultaneous requests, what causes a fast growth of times since 160$^{\text{th}}$ service. Moreover, Pastry updates the leaf sets to maintain the coherence of the system, and may be for this reason the simultaneous version times reach 320 s. Whereas in the sequential version, the bootstrap receives only one request each minute; the update of the leaf sets and routing tables is, thus, recovered so that we got reduced times (2 s maximum). Our

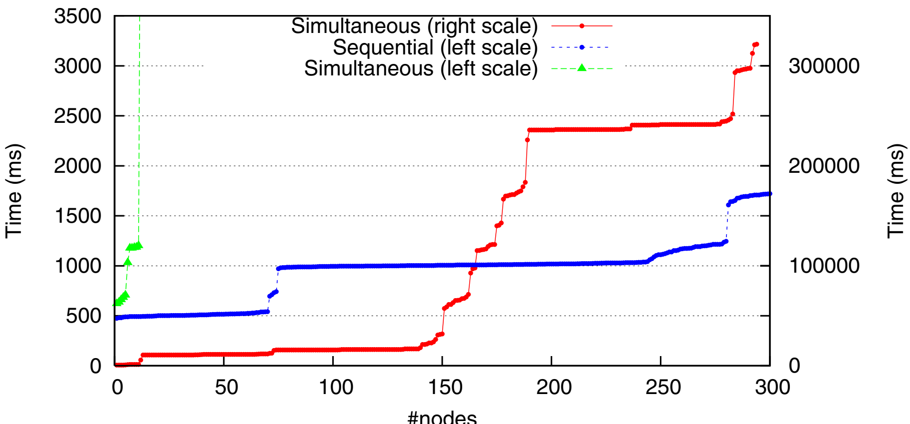


Fig. 2. Elapsed time for registrations of Pastry services

measurements show that it is necessary to create a second bootstrap to initialize a new ring when the number of simultaneous registrations overtake 160 services.

4.4 Synthesis

The comparison of the three libraries (Bonjour, Avahi and Pastry) from the viewpoint of simultaneous registration times of approximately 300 services on the platform Grid'5000 (one service by machine) shows that Avahi is the best since it spends less time (last registered service needs 1000 ms). Bonjour requires 1300 ms moreover to register the last service. Pastry gives times close to those given by Avahi until 160 registrations, beyond this, it spends times definitely larger (until 320 000 ms) that those of Avahi and Bonjour. When we sequentially register one service on each machine (we register about 300 services), we can mention that there is not a great difference between the three libraries. In fact, Bonjour and Avahi give similar results. Pastry spends almost same time to register 60% of services, needs less times to register first 30% but more times than Avahi and Bonjour for the rest (10%).

5 Performance of Discovery Services

The second metric is to measure the necessary time to browse a registered service. Then for each system (Bonjour, Avahi and Pastry) we measure the elapsed time between the registration end and the discovery time. We repeat the same benchmarks for both simultaneous and sequential registration. For that, we dedicate one machine which runs the browser to discover services.

5.1 Discovery Behavior of Bonjour

Bonjour proves a good performance in discovering services. In fact, it is able to discover 307 services registered on 307 machines (one service on one machine). Furthermore, the discovery time doses not exceed 1 second. That leads us to affirm that the implementation by Apple of DNS-SD functions is good and gives satisfactory results.

5.2 Discovery Behavior of Avahi

As we have already mentioned, Avahi uses avahi-daemon and executes a request via D-Bus to publish a service by a multicast package. The machine which launches the discovery program (a browser program which remains listening to services) loses 60% of simultaneously registered services. Moreover, the discovery time increases beyond 49 registrations to reach 900 s in the registration of the 73th service. Beyond that, the discovery program spends around 220 s to discover a registered service (see figure 3). Contrarily to the simultaneous registration, when we register the services in a sequential manner (for instance each minute), the Browser is able to discover more services (303 among 307 registered services). In addition, the discovery time is better (maximum 4 s) for 204 services except

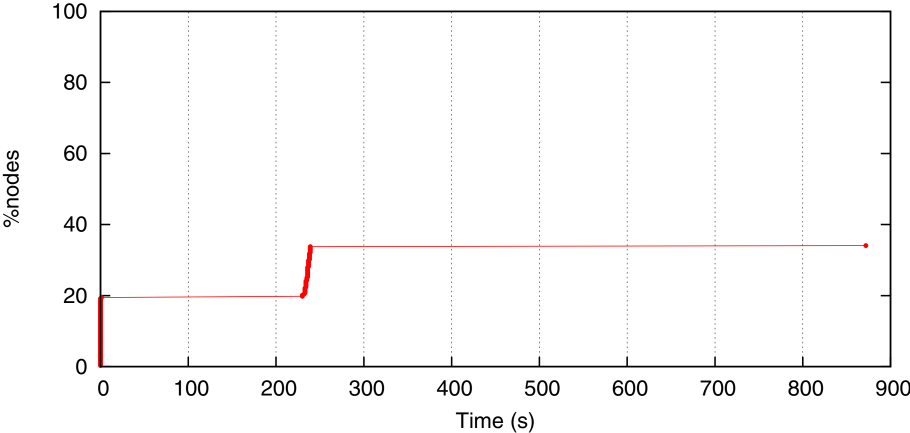


Fig. 3. Elapsed time for browsing simultaneously registered Avahi services

that 9 services need intervals of time of 2835-8686 s. The browser, which has the avahi-daemon running on it, is unable to receive, at a given moment, all multicast sends emitted by the nodes which registered the services at the same time (simultaneous case). This can explain the loss of services in the simultaneous version. Whereas, in sequential version the browser needs more time to discover the four lost services.

5.3 Discovery Behavior of Pastry

In both types of sequential and simultaneous registrations, pastry provides good discovery times (1 s maximum). Meanwhile, the number of discovered services for simultaneous registrations, the browser discovers 270 among 293 services, whereas for sequential registration, the browser discovers 275 of the 292 services what corresponds to a light improvement in comparison to the previous case. That can leads us to affirm that the Browser fails to recover all services publications notifications when registrations exceed 270.

6 Related Work

Experiments and analysis of P2P networks have been conducted over the Grid'5000 platform for the generic JXTA P2P framework [16]. In this article, the goal of the performed benchmarks is similar to our goal. It concerns to answer common and unanswered questions: how many rendezvous peers are supported by JXTA in a given group and what is the expected time to discover resources in such groups?

Two main protocols of JXTA have been evaluated in [16]: 1) the peer-view protocol used to organize super peers, known as rendezvous peers, in a JXTA overlay and 2) the discovery protocol, that relies on the peer-view protocol,

used to find resources inside a JXTA network. All sites of Grid'5000 were used and a mix of hundreds of rendezvous and normal peers, called edge peers, have been deployed on at most 580 nodes. Results show that with default values for parameters of the peer-view protocol, the goal of the algorithm is not achieved, even with just 45 rendezvous peers. However, parameter tuning makes it possible to reach larger configurations in terms of number of rendezvous peers. For the discovery protocol, authors show that discovery time is rather smaller, provided that all rendezvous peers satisfy a given property. These results give developers a better view of the scalability of JXTA protocols.

Our results[1] augmented with those of [16] clearly demonstrate that for open source projects as well as for industrial software with production quality, there is a strong need to test and evaluate the properties of the distributed system in real large platform such as Grid'5000.

7 Conclusion and Future Work

With the growth of grids size, it is feasible to use P2P systems known for their scalability and the management of high volatility. In this context, we studied in this paper three protocols for resource discovery which are Bonjour, Avahi and Pastry. The three protocols showed high performances except that Avahi failed to discover all services in simultaneous version and Pastry spends a considerably long time to register all services simultaneously. We are going to continue working on the three protocols. Indeed, Bonjour is very powerful in registration and discovery in both sequential and simultaneous versions. Pastry also proved its performance in sequential registration and also in simultaneous case provided when we does not exceed 160 registrations at a given moment, what does not represent really a weakness point if we do not work with, let us say 10 million nodes. Avahi has the advantage of being free with accessible source code, which makes its study much easier than Bonjour. As technical point of view, the ZeroConf API does not offer full functionalities to build grid middleware, whereas Pastry offers an open source API developed with Java containing the preliminary functions necessary to the development of this middleware. Our final objective is to build a Desktop Grid middleware based on one of these protocols.

As mentioned before, the initial question was: "how to decentralize the services offered by Desktop Grids?". Our idea is to capitalize on existing systems (for instance on XtremWeb [17]) rather than inventing a completely new one. We are currently implementing a prototype according to the following vision: a user requests for a computation. He provides a tasks graph and codes implementing his distributed application. The idea is to deploy locally a master node (according to the XtremWeb terminology) and to request for participants. Negotiations to select them should now take place. The Publish/Subscribe infrastructure is used to solve the problem. For instance, each node multicasts, periodically, its state (idle, slave, master) and also information about its local load or its use cost, in

[1] For more results see the LIPN internal report on this URL: https://hal.ccsd.cnrs.fr/docs/00/15/93/88/PDF/acdj.pdf

order to provide metrics for choosing the participants. Under these assumptions, the master node can select a subset of slave nodes according to a strategy that could balance the "power" of the node and the "price" to use it. When a master node finishes, it becomes free and returns to the sleeping state or, if it needs to start a new job (because the user has many works to do) it is ready to accept it. When a slave node finishes its tasks (as a slave), it has the possibility to become a master or a slave node again but not necessarily for the same master. In this way, we ensure an automatic load balancing schema and the whole system becomes less centralized. Again, the key idea is to rely on existing Desktop Grid middleware but also to control and coordinate multiple instances through Publish/Subscribe systems.

Acknowledgement

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners. We would like also to thank Mathieu Jan for his fruitful comment on an early version of this paper.

References

1. Teragrid, http://www.teragrid.org/
2. Enabling grids for escience, http://public.eu-egee.org/
3. Osg, http://www.opensciencegrid.org/
4. Grid'5000, http://www.grid5000.fr
5. Rezmerita, A., Morlier, T., Néri, V., Cappello, F.: Private virtual cluster: Infrastructure and protocol for instant grids. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 393–404. Springer, Heidelberg (2006)
6. Instant grid, http://www.lri.fr/~rezmerit/Instant%20Grid.html
7. Steinberg, D., Cheshire, S.: Zero Configuration Networking: The Definitive Guide, 1st edn. O'Reilly Media, Inc., Sebastopol (2005)
8. ZeroConfguration, http://www.zeroconf.org
9. Avahi, http://www.avahi.org
10. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware 2001: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, London, UK, pp. 329–350. Springer, Heidelberg (2001)
11. FreePastry, http://www.freepastry.org
12. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: SIGCOMM 2001: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 161–172. ACM Press, New York (2001)
13. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM 2001: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 149–160. ACM Press, New York (2001)

14. OAR, http://gforge.inria.fr/projects/oar/
15. KADeploy, http://gforge.inria.fr/projects/kadeploy/
16. Antoniu, G., Cudennec, L., Duigou, M., Jan, M.: Performance scalability of the JXTA P2P framework. In: Proc. 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, CA, USA, March 2007, pp. 108–131 (2007)
17. Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V., Lodygensky, O.: Computing on large scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. Future Generation Computer Systems 21(3), 417–437 (2005)

Towards a Security Model to Bridge Internet Desktop Grids and Service Grids

Gabriel Caillat[1], Oleg Lodygensky[1], Etienne Urbah[1],
Gilles Fedak[2], and Haiwu He[2]

[1] Laboratoire de l'Accelerateur Lineaire,
Universite Paris Sud Bat 200 , 91 Orsay, France
{gcaillat,lodygens,urbah}@lal.in2p3.fr
[2] INRIA Saclay, LRI,
Universite Paris Sud Bat 490 , 91 Orsay, France
{fedak,haiwu.he}@lri.fr

Abstract. Desktop grids, such as XtremWeb and BOINC, and service grids, such as EGEE, are two different approaches for science communities to gather huge computing power from a large number of computing resources. Nevertheless, little work has been done to combine these two Grids technologies in order to establish a seamless and vast grid resource pool. In this paper we address the security issues when bridging Service Grid with Desktop Grid. We first present how to bridge EGEE resources with our XtremWeb platform using the gliding-in mechanism. Then we describe a new Desktop Grid security model to bridge this anonymous environment to the strongly securized Service Grid one. Finally we describe an implementation of this security model in the XtremWeb middleware and report on performance evaluation.

Keywords: Desktop Grid, Service Grid, BOINC, XtremWeb, Security.

1 Introduction

There is a growing interest among scientific communities to use Grid computing infrastructures to solve their grand-challenge problems and to further enhance their applications with extended parameter sets and greater complexity. Researchers and developers in Service Grids *(SG)* first create a Grid service that can be accessed by a large number of users. A resource can become part of the Grid by installing a predefined software set, or middleware. However, the middleware is usually so complex that it often requires extensive expert effort to maintain. It is therefore natural, that individuals do not often offer their resources in this manner, and SGs are generally restricted to larger institutions, where professional system administrators take care of the hardware/middleware/software environment and ensure high-availability of the Grid. Examples of such infrastructures are EGEE, the NorduGrid, or the NGS (National Grid Service) in the UK. Even though the original aim of enabling anyone to join the Grid with one's resources has not been fulfilled, the largest Grid in the world (EGEE) contains

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 247–259, 2009.
© Springer-Verlag Berlin Heidelberg 2009

around forty thousand processors. Anyone who obtains a valid certificate from a Certificate Authority (CA) can access those Grid resources that trust that CA. This is often simplified by Virtual Organization (VO) or community authorization services that centralizes the management of trust relationships and access rights.

Desktop Grids *(DG)*, literally Grids made of Desktop Computers, are very popular in the context of "Volunteer Computing" for large scale "Distributed Computing" projects like SETI@home [ACK+02] and Folding@home. They are very attractive, as "Internet Computing" platforms, for scientific projects seeking a huge amount of computational resources for massive high throughput computing. DG uses computing, network and storage resources of idle desktop PCs distributed over multiple LANs or the Internet. Today, this type of computing platform aggregates one of the the largest distributed computing systems, and currently provides scientists with tens of TeraFLOPS from hundreds of thousands of hosts. In DG systems, such as BOINC [And04] or XtremWeb [FGNC01], anyone can bring resources into the Grid, installation and maintenance of the software is intuitive, requiring no special expertise, thus enabling a large number of donors to contribute into the pool of shared resources. On the downside, only a very limited user community (i.e., target applications) can effectively use DG resources for computation. For instance the BOINC project features less than 50 applications, and the top 5 projects share more than 50 % of the total compute power. Because users are Internet volunteers, there cannot be security model based on trust between users. Because of users anonymity, security solution for DG relies on autonomous mechanism such as sandbox or result certification to prevent attacks from other users. As a consequence, DG systems are not yet ready to be integrated in complex Grid infrastructure which requires a high level user right management, authentication, authorization and rights delegation.

Until now, these two kinds of Grid systems are completely separated and there is no way to exploit their individual advantageous features in a unified environment. However, with the objective to support new scientific communities that need extremely large numbers of resources they can't find in SG, the solution could be to interconnect these two kinds of Grid systems into an integrated Service Grid–Desktop Grid (SG–DG) infrastructure. Our research are part of the work conducted by a new European FP7 infrastructure project : EDGeS (Enabling Desktop Grids for e-Science) [CMEM+08], which aims to build technological bridges to facilitate interoperability between DG and SG.

In this paper, we describe the research issues on how such an integrated SG–DG infrastructure can be established from a security point of view. We first review the security model of existing approaches to bridge the two classes of Grid systems. Our technical contribution is two folds. First, we propose a new users classification into Desktop Grids which allow to manage anonymous DG users and SG users in a global SG-DG infrastructure. We detail expected modifications in our XtremWeb middleware. Second, we present a bridge between the EGEE

grid and XtremWeb based on the *gliding in* solution. We show that this solution provides high security level, fault-tolerance, performance and scalability.

In the next section, we give an overview of the Service Grid and the Desktop Grid security model. In Section 3, we describe the existing solution to bridge Service Grid and Desktop Grid. In Section 4, we present our security architecture and its implementation within the XtremWeb Desktop Grid and the integration in the EGEE Grid. Performance evaluation is shown in Section5. In Section 6, we present our concluding remarks.

2 Background

2.1 Security Model of Desktop Grids

In this section we review the security model of several Desktop Grid systems.

The BOINC [And04] middleware is a popular Volunteer Computing System which permits to aggregate huge computing power from thousands of Internet resources. A key points is the asymmetry of its security model : there are few projects well identified and which belongs to established institutions (Univ. of Berkeley, Univ of Haifa...) while volunteers are numerous and anonymous. The notion of users exists in BOINC, which aims to manage volunteers contributions. However, this user definition is close to the one of avatar : it allows users to participate to forum and to receive credits according to the computing time and power given to the project.

Despite anonymity, the security model is based on trust. Volunteers trust the project they are contributing to. Security mechanism is simple and based on asymmetric cryptography.

Security model aims at enforcing the trust between volunteers and the project itself. At installation time, the project owners produce a pair of public/private keys and store them in a safe place, typically in a machine isolated from the network, as recommended on the BOINC web site. When volunteers contribute for the first time to the project, they obtain the public key of the project. Project owners have to digitally sign the project application files, so that volunteers can verify that the binary codes downloaded by the BOINC client really belongs to the project. This mechanism ensures that, if a pirate get access to one of the BOINC server, he would not be able to upload malicious code to hundreds of thousands resources. If volunteers trust the projects, the reverse is not true. To protect against malicious users, BOINC implements a result certification mechanism [Sar02], based on redundant computation. BOINC gives the ability to project administrator to write their own custom results certifying code according to their application.

XtremWeb is an Internet Desktop Grid middleware which also permits public resources computing. It differs from BOINC by the ability given to every participants to submit new application and tasks in the system. XtremWeb is a P2P system in the sense that every participants can provide computing resources but also utilize others participants' computing resources. XtremWeb is organized as a three-tiers architecture where clients consumes resources, worker provides

resources and coordinator is a central agent which manages the system by performing the scheduling and fault-tolerance of tasks. XtremWeb implements the notion of user, used to facilitate the platform management and to separate between users' tasks, results and applications. In contrast with BOINC, because everyone can submit application, there cannot be any form of trust between users, application, results and even the coordinator itself. Thus XtremWeb security model is based on autonomous mechanisms which aims at protecting each components of the platform from the others elements. For instance, to protect volunteers' computer from malicious code, a sandbox mechanism is used to isolate and monitor the running application, and prevent it to damage volunteers system. Public/private keys mechanism are also used to authenticate the coordinator to prevent results to be uploaded to an other coordinator.

The Xgrid system, proposed by Apple is a Desktop Grid designed to run on a local network environment. Xgrid features ease of use and ease of deployment. To work, the Xgrid system needs a Xgrid server, which can be configured with or without password. If the server run without password, then every user in the local environment can submit jobs and application; otherwise a password is needed to do so. Computing nodes, in the Xgrid system can accept jobs or no, this property is set on the computing nodes itself. Thus there is no real distinction between users and there's no possibility for a user or a machine to accept or refuse other users' application or work. While this solution is acceptable when used within a single organization (lab or small company), this solution won't scale to a Service Grid setup which typically aims at several institutions to cooperate.

2.2 Security Model of Service Grids

The Grid Security Infrastructure (GSI) in EGEE enables secure authentication and communication over an open network. GSI is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol, with extensions for single sign-on and delegation. In order to authenticate himself to Grid resources, a user needs to have a digital X.509 certificate issued by a Certification Authority (CA) trusted by EGEE; Grid resources are generally also issued with certificates to allow them to authenticate themselves to users and other services. The user certificate, whose private key is protected by a password, is used to generate and sign a temporary certificate, called a proxy certificate (or simply a proxy), which is used for the actual authentication to Grid services and does not need a password. As possession of a proxy certificate is a proof of identity, the file containing it must be readable only by the user, and a proxy has, by default, a short lifetime (typically 12 hours) to reduce security risks if it is stolen. A user needs a valid proxy to submit jobs; those jobs carry their own copies of the proxy to be able to authenticate with Grid services as they run (so that the job can access user data, for example). For long-running jobs, the job proxy may expire before the job has finished, causing the job to fail. To avoid this, there is a proxy renewal mechanism to keep the job proxy valid for as long as needed. In terms of security a proxy is a compromise. Since the private key is sent with it anyone who steals it can impersonate the owner, so proxies need

to be treated carefully. Also there is no mechanism for revoking proxies, so in general even if someone knows that one has been stolen there is little they can do to stop it being used. On the other hand, proxies usually have a lifetime of only a few hours so the potential damage is fairly limited.

Grid security is based on the concept of public key encryption. Each entity (user, server...) has a private key which must be kept totally secure.

Each private key has its associated public key; they are referred as asymmetric keys. Any encryption using one key can be decrypted using the associated one. This mechanism is used to prove identity, to encrypt data and to check their integrity.

Certificates are issued by a Certification Authority (CA) which has its own certificate.

To check the validity of a certificate, the public key of the CA is then needed. Potentially this could create an infinite regression, but this is prevented by the fact that CA certificates, known as root certificates, are self-signed, i.e. the CA signs its own certificate.

A system called VOMS (VO Management Service) is used in EGEE to manage information about the roles and privileges of users within a VO. This information is presented to services via an extension to the proxy. At the time the proxy is created one or more VOMS servers are contacted, and they return a mini certificate known as an Attribute Certificate (AC) which is signed by the VO and contains information about group membership and any associated roles within the VO.

To create a VOMS proxy the ACs are embedded in a standard proxy, which is signed with the private key of the parent certificate. Services can then decode the VOMS information and use it as required, e.g. a user may only be allowed to do something if he has a particular role from a specific VO. One consequence of this method is that VOMS attributes can only be used with a proxy, they cannot be attached to a CA-issued certificate.

3 Bridging Service Grids and Desktop Grids

There exists two main approaches to bridge SG and DG (see Fig. 1). In this section we present the principles of these two approaches and discuss them according to security perspective.

The superworker approach. The *superworker*, proposed by the Lattice [MBC08] project and the SZTAKI Desktop Grid [BGK+07], is a first solution. This enables the usage of several Grid or cluster resources to schedule DG tasks. The superworker is a bridge implemented as a daemon between the DG server and the SG resources. From the DG server point of view, the Grid or cluster appears as one single resource with large computing capabilities. The superworker continuously fetches tasks or work units from the DG server, wraps and submit the tasks accordingly to the local Grid or cluster resources manager. When computations are finished on the SG computing nodes, the superworker sends back

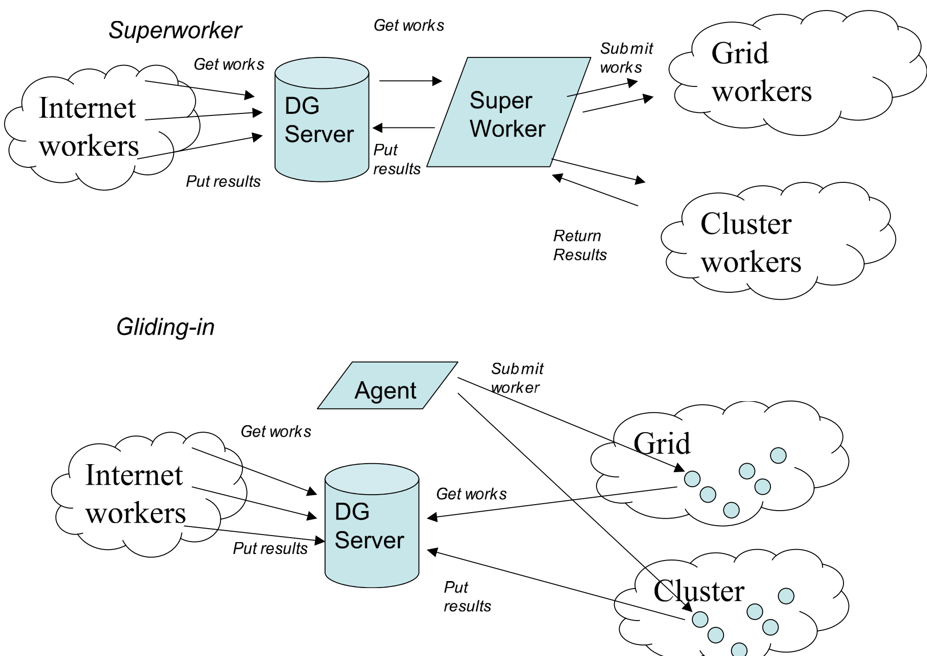


Fig. 1. Bridging Service Grid and Desktop Grid, the superworker approach versus the Gliding-in approach

the results to the DG server. Thus, the superworker by itself is a scheduler which needs to continuously scan the queues of the computing resources and watch for available resources to launch jobs.

Since the superworker is a centralized agent this solution has several drawbacks : *i*) the superworker can become a bottleneck when the number of computing resources increases, *ii*) the round trip for a work unit is increased because it has to be marshalled/unmarshalled by the superworker, *iii*) it introduces a single point of failure in the system, which has low fault-tolerance. On the other hand, this centralized solution provides better security properties, concerning the integration with the Grid. First the superworker does not require modification of the infrastructure, it can be run under any user identity as long as the user has the right to submit jobs on Grid. Next, as works are wrapped by the superworker, they are run under the user identity, which conforms with the regular security usage, in contrast with the approach described in the following paragraph.

The Gliding-in approach. The *Gliding-in* approach to cluster resources spread in different Condor pool using the Global Computing system (XtremWeb) was first introduced in [LFC+03]. The main principle consists in wrapping the XtremWeb worker as regular Condor task and in submitting this task to the Condor pool. Once the worker is executed on a Condor resource, the worker

pulls jobs from the DG server, executes the XtremWeb task and return the result to the XtremWeb server. As a consequence, the Condor resources communicates directly to the XtremWeb server. Similar mechanisms are now commonly employed in Grid Computing [TL04]. For example Dirac [TGSR04] uses a combination of push/pull mechanism to execute jobs on several Grid clusters. The generic approach on the Grid is called a *pilot job*. Instead of submitting jobs directly to the Grid gatekeeper, this system submits so-called pilot jobs. When executed, the pilot job fetches jobs from an external job scheduler.

The gliding-in or pilot job approach has several advantages. While simple, this mechanism efficiently balance the load between heterogeneous computing sites. It benefits from the fault tolerance provided by the DG server; if Grid nodes fail then jobs are rescheduled to the next available resources. Finally, as the performance study of the Falkon [RZD+07] system shows, it gives better performances because series of jobs do not have to go through the gatekeeper queues which is generally characterized by long waiting time, and communication is direct between the worker running on the computing element (*CE*) and the DG server without intermediate agent such as the superworker. From the security point of view, this approach breaks the Grid security rule about Pilot Jobs. This rule does not allow actual jobs owner to be different than pilot job owner. This is a well known issue of pilot jobs and new solution such as *gLExec* [SKV+07] are proposed to circumvent this security hole.

4 Bridging XtremWeb to EGEE

Our goal is to safely aggregate EGEE worker nodes in a single XtremWeb network. In this network, we assume that the users connect to the dispatcher administration domain to submit tasks. XtremWeb has the responsibility to ensure user authentication, hosts (workers) integrity, application and results protection and user execution logging.

In the rest of the paper, we based our study of the security model on the gliding-in technology. XtremWeb Workers are submitted to EGEE computing elements using JSDL wrappers.

4.1 User Authentication and Execution Logging

The coordinator site manages a list of authorized users as ACLs. It is the responsibility of the system administrator to register new users (and revoke non desired ones) on the coordinator. After registration, the coordinator provides a key to be used by the user for each subsequent connection. For each connection, a challenge is ran in order to ensure that the user is registered on the coordinator. All communications between the user XW client and the coordinator are encrypted using SSL. Then the coordinator works as a proxy for the user: all tasks are submitted to the workers through the coordinator credential. All executions on the workers are logged in the security perspective: all tasks contain a descriptor with the actual user credential so that workers and coordinator can take appropriate corrective action (user revocation), in case of security problem.

The design does not currently rely on certificates and presents a certain degree of risk for "Man is the Middle" (*MIM*) attacks but risks are very limited since 1) attacks should origin from within EGEE subclusters only (due to TCP protocols), and 2) workers and clients software include coordinator public key, then if one is able to securely ensure worker and client binaries installation to dedicated pools, the full system is not subject to *MIM* attacks since key exchanges will not be necessary any more.

A certification system based on X.509 certificates is under integration in XtremWeb. Subsequent experiments and futures XtremWeb installations will implement one, based on Open-SSL, allowing extension of clients and workers authentication by the coordinator.

4.2 Applications, Parameters and Results Protection

EGEE subclusters belonging to different administration domains fetch applications and tasks, and store results on the central coordinator. The only security issue concerning applications, parameters and results transfers is then about the connections between EGEE worker nodes and the coordinator. To ensure connection security between domains, 1) every connection from any client and worker to the coordinator is encrypted through *SSL* tunnels; 2) workers can only connect to the coordinator for which they have the public key. These two mechanisms prevent malicious participants to be able to intercept and read any connection, to connect to the coordinator and EGEE worker nodes to connect to a wrong XtremWeb coordinator.

4.3 Node Integrity

If, for any reason, a malicious user succeeds on accessing the system and launching an aggressive application, XtremWeb workers still protect their host by implementing *sand-boxing*[AKS99, AR99, GWTB96] for binary applications. This is a secure way to execute applications, providing rights to do some actions and denying some others. One should note that Java applications are always executed inside a virtual machine which includes security[GMPS97]; XtremWeb uses this functionality in two levels, one for the worker itself and a more restrictive one for the downloaded Java byte code. On the contrary, binary (or *native*) applications have access to the full hosting system by nature; workers are configured to run any task of that type inside a sand-box which is fully customizable, from memory usage to file system operations.

Java and sand-boxes, have performance costs[BSPF01]; one can then disable this functionality on highly secured systems, such as clusters under a fully closed firewall.

4.4 Access Confinement

XtremWeb 2.0.0 introduces mechanisms aimed to secure and confine distributed resources usage; this is done thanks to the notion of user and access rights. These

new features permit to extend user actions over the platform as well as to secure resource usage and confine application deployment.

Access rights must be understood as linux file system ones (e.g. 0x755, uog+r etc.) and are used to define data (which is also a new paradigm in XtremWeb 2.0.0 but not discussed in this deliverable), application and job accesses. The default access is *0x755* which grants full access (read, write, execute) to owner, and limited access (read, execute) to users belonging to the owner group, as well as other users. Denying access to non group users, for example, consists to set access rights to *0x750*. The middleware includes the *xwchmod* command to modify access rights.

Any user can insert its own applications in the platform. This feature could lead to security hazards since this could allow users to insert malicious applications. This is solved by access rights. A *standard* user can only insert *private* applications; any submitted jobs referring private applications are private too. There is no way to modify this; even *xwchmod* cannot help to modify access rights of private entities (applications or jobs). A private entity has its access rights set to *0x700* which grants owner access only. A private entity will only be managed by private workers which are described below.

Inserting group (i.e. access rights *0x750*) entities needs advanced privileges. All users belonging to the group can access group entities. Submitting a job for a group application creates group jobs (i.e. access rights *0x750*). A group job can only be run by group workers which are described below.

Finally, inserting public (i.e. access rights *0x755*) entities needs advanced privileges too. All users can submit jobs for public applications. This creates public jobs (i.e. access rights *0x755*). A public job can only be run by public workers which are described below.

User rights, associated with access rights, permit to define public, groups and private levels, which grant allowed user actions. The three main level rights are *standard*, *worker* and *advanced*. The *standard* level grants data management, job submission and private application insertion as defined earlier. The *advanced* level grants full access to the platform including private, group and public entities, as well as users, user groups and workers management.

To understand the *worker* right level, one must understand that workers run using a registered identity. When a worker communicates with the coordinator, it presents its identity. This identity defines user rights, among others.

The *worker* level right defines *public* workers. This level right permits to delegate user rights to those public workers so that they can access entities as if they were the entity owner. A public worker can bypass entities access rights in order to update them even if their access rights do not allow that action. This is used to update jobs status to *COMPLETED* when it has successfully been computed, or to store jobs results and set results owner to the job owner. That last can then download its results. At installation time, the platform includes a specific user defined with worker level right, aiming to deploy public workers. Workers can also be group ones. Group workers are public ones restricted to a group. They use an identity belonging in a group, with *worker* user rights level.

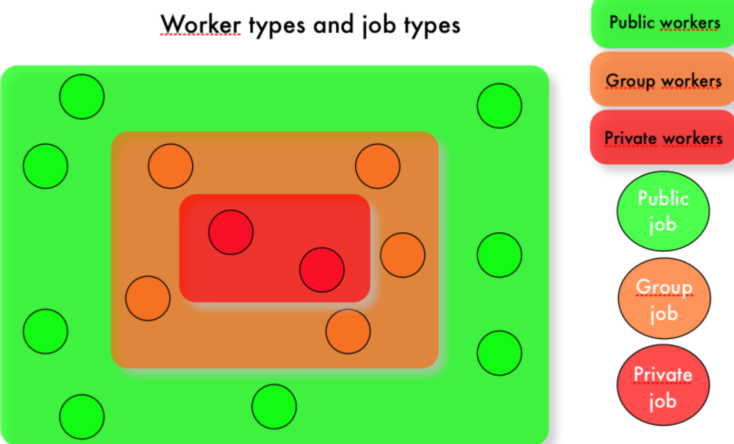


Fig. 2. Worker levels and jobs levels

Group workers will compute jobs submitted by any user of their group only. Finally, workers can be private. Private workers are identified whithout *worker* user rights level. They can only compute private jobs.

The Figure 2 summarizes worker levels and the types of job they can run. We see that private jobs are run on private workers only, groups jobs on group workers only, and public jobs on public workers only.

5 Performance Evaluation

To demonstrate the full system, we ran an application over our SG/DG platform. The application consists in a multi-parameters computation requiring a large set of independent tasks. We submitted 185 tasks XtremWeb aggregating volunteers resources and EGEE worker nodes.

Table 1 summarizes resource aggregated from XtremWeb and from EGEE.

Table 2 details these resource and shows four different types according to CPU speed.

Figure 3 shows a good distribution of the jobs among resources; differences comes from availability and resource types themselves. We can see that some resources have only computed one job; this is because the bridge configures

Table 1. Available resources per platform

	XtremWeb	EGEE	Total
Linux	1	10	11
Win32	1	0	1
Mac OS X	32	0	32
Total	34	10	44

Table 2. Types of available resources

OS	CPU	CPU speed	Total
`Linux`	AMD64	2.3GHz	11
`Win32`	ix86	2.0GHz	1
`Mac OS X`	ix86	2.0GHz	1
`Mac OS X`	ix86	2.0GHz	10
`Mac OS X`	PPC	1.0GHz	5
`Mac OS X`	PPC	1.5GHz	6
`Mac OS X`	PPC	2.0GHz	10
`Total`	3	4	44

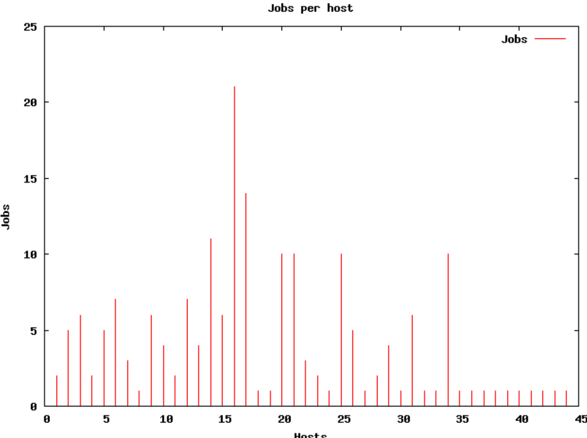


Fig. 3. Jobs per host in a SG/DG platform

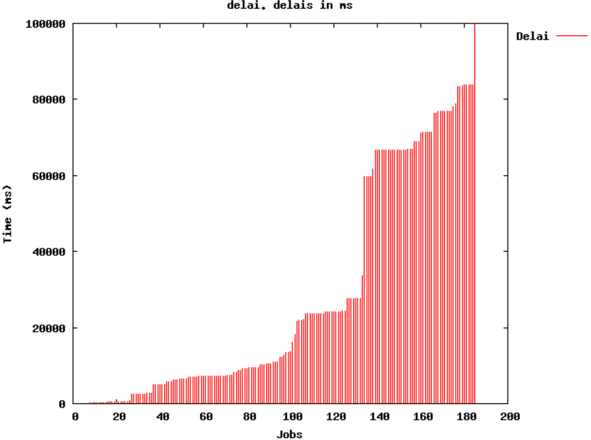


Fig. 4. Jobs execution delays

resources aggregated from SG to run a single job. This ensures that DG applications do not overload SG resources.

The impact of the resource heterogeniety is even more visible on figure 4 which presents the job execution time, sorted in increasing order.

6 Conclusion

Bridging Service Grids and Desktop Grids raises many issues. To enable actual infrastructure, gathering both Grid and Internet users, application, computing and storage resources requires new security model. In this paper we have reviewed the security requirements of DG and SG. The discussion about bridging technologies convince us to select the gliding-in solution even if it stresses the security requirements. Thus we have proposed a security model which distinguishes between *anonymous* users who are typically Internet volunteers and *certified* users who are the Grid users with a valid X.509 certificate delivered by the EGEE author. We have implemented this security model within the XtremWeb framework and showed that, when integrated with the EGEE infrastructure, this ensures a high security level for both the Grid and the volunteers' PC.

Acknowledgments

The EDGeS (Enabling Desktop Grids for e-Science) project receives Community funding from the European Commission within Research Infrastructures initiative of FP7 (grant agreement Number 211727).

References

[ACK+02] Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: Seti@home: an experiment in public-resource computing. Commun. ACM 45(11), 56–61 (2002)

[AKS99] Alexandrov, A., Kmiec, P., Schauser, K.: Consh: a confinied execution environment for internet computations. In: Proceedings of the Usenix annual technical conference (1999), http://www.usenix.org/events/usenix99/

[And04] Anderson, D.: BOINC: A System for Public-Resource Computing and Storage. In: Proceedings of the 5th IEEE/ACM International GRID Workshop, Pittsburgh, USA (2004)

[AR99] Acharya, A., Raje, M.: Mapbox: using parameterized behavior classes to confine applications. Technical report TRCS99-25, University of California, Santa Barbara (1999)

[BGK+07] Balaton, Z., Gombas, G., Kacsuk, P., Kornafeld, A., Kovacs, J., Marosi, A.C., Vida, G., Podhorszki, N., Kiss, T.: Sztaki desktop grid: a modular and scalable way of building large computing grids. In: Proc. of the 21th International Parallel and Distributed Processing Symposium, Long Beach, California, USA, March 26-30 (2007)

[BSPF01] Bull, J.M., Smith, L.A., Pottage, L., Freeman, R.: Benchmarking java against c and fortran for scientific applications. In: ISCOPE Conference. LNCS, vol. 1343, pp. 97–105. Springer, Heidelberg (2001)

[CMEM+08] Cardenas-Montes, M., Emmen, A., Marosi, A.C., Araujo, F., Gombas, G., Terstyanszky, G., Fedak, G., Kelley, I., Taylor, I., Lodygensky, O., Kacsuk, P., Lovas, R., Kiss, T., Balaton, Z., Farkas, Z.: Edges: bridging desktop and service grids. In: Proc. of the 2nd Iberian Grid Infrastructure Conference, University of Porto, Portugal, May 12-14 (2008)

[FGNC01] Fedak, G., Germain, C., Néri, V., Cappello, F.: XtremWeb: A Generic Global Computing Platform. In: Proceedings of 1st IEEE International Symposium on Cluster Computing and the Grid CCGRID'2001, Special Session Global Computing on Personal Devices, Brisbane, Australia, May 2001, pp. 582–587. IEEE/ACM (2001)

[GMPS97] Gong, L., Muller, M., Prafullchandra, H., Schemers, R.: Going beyond the sandbox: an overview of the new security architecture in the java development kit 1.2. In: Usenix Symposium on Internet Technologies and Systems (1997)

[GWTB96] Goldberg, I., Wagner, D., Thomas, R., Brewer, E.: A secure environment for untrusted help application – confining the wily hacker. In: Proceedings of the 6th Usenix Security Symposium (1996)

[LFC+03] Lodygensky, O., Fedak, G., Cappello, F., Neri, V., Livny, M., Thain, D.: XtremWeb & Condor: Sharing Resources Between Internet Connected Condor Pools. In: Proceedings of CCGRID 2003, Third International Workshop on Global and Peer-to-Peer Computing (GP2PC 2003), Tokyo, Japan, pp. 382–389. IEEE/ACM (2003)

[MBC08] Myers, D.S., Bazinet, A.L., Cummings, M.P.: Grids for Bioinformatics and Computational Biology. In: Expanding the reach of Grid computing: combining Globus- and BOINC-based systems. Wiley Book Series on Parallel and Distributed Computing (2008)

[RZD+07] Raicu, I., Zhao, Y., Dumitrescu, C., Foster, I., Wilde, M.: Falkon: a fast and light-weight task execution framework. In: IEEE/ACM SuperComputing (2007)

[Sar02] Sarmenta, L.F.G.: Sabotage-Tolerance Mechanisms for Volunteer Computing Systems. Future Generation Computer Systems 18(4), 561–572 (2002)

[SKV+07] Sfiligoi, Koeroo, O., Venekamp, G., Yocum, D., Groep, D., Petravick, D.: Addressing the Pilot security problem with gLExec. Technical Report FERMILAB-PUB-07-483-CD, Fermi National Laboratory (2007)

[TGSR04] Tsaregorodtsev, A., Garonne, V., Stokes-Rees, I.: Dirac: A scalable lightweight architecture for high throughput computing. In: Fifth IEEE/ACM International Workshop on Grid Computing, GRID 2004 (2004)

[TL04] Thain, D., Livny, M.: Building reliable clients and services. In: Foster, I., Kesselman, C. (eds.) The Grid: Blueprint for a New Computing Infrastructure, 2nd edn. Morgan Kaufman, San Francisco (2004)

Workshop on Productivity and Performance (PROPER 2008)

The current and future increase of computer performance is driven by raising the number of cores on a single chip and within a supercomputer. As a consequence, applications need to harness much higher degrees of parallelism in order to satisfy their growing demands for computing power. Writing code that runs correctly and efficiently on large numbers of processors and cores is extraordinarily challenging. The increased concurrency levels also place higher demands on the development process and thus require adequate and scalable tool support for debugging and performance analysis. This workshop serves as a forum for discussing methods and experiences related to scalability and usability of HPC programming tools, including integration with compilers and the overall development environment.

The workshop topics include tools and tool approaches for parallel program development and analysis, in particular focusing on high-performance computing (HPC). Furthermore, it covers debugging and correctness checking tools for this area as well as approaches and tools for performance measurement and evaluation. Last but not least, it presents success stories of optimization of parallel scalability and performance achieved using tools.

With these topics, the PROPER workshop wants to bring together research with experimental to product level tools in the areas of interest as well as the tool developers with the tool users.

Matthias Müller
Andreas Knüpfer

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, p. 261, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Enabling Data Structure Oriented Performance Analysis with Hardware Performance Counter Support*

Karl Fürlinger[1,2], Dan Terpstra[2], Haihang You[2], Phil Mucci[2], and Shirley Moore[2]

[1] Computer Science Division,
EECS Department,
University of California at Berkeley
fuerling@EECS.Berkeley.EDU
[2] Innovative Computing Laboratory,
Department of Electrical Engineering and Computer Science,
University of Tennessee
{karl,terpstra,you,mucci,shirley}@eecs.utk.edu

Abstract. An interesting and as of yet under-represented aspect of program development and optimization are data structures. Instead of analyzing data with respect to code regions, the objective here is to see how performance metrics are related to data structures. With the advanced performance monitoring unit of Intel's Itanium processor series such an analysis becomes possible. This paper describes how the hardware features of the Itanium 2 processor are exploited by the perfmon and PAPI performance monitoring APIs and how PAPI's support for address range restrictions has been integrated into an existing profiling tool to achieve the goal of data structure oriented profiling in the context of OpenMP applications.

1 Introduction

Parallel performance analysis is traditionally the domain of scientific application developers. With the increasingly widespread adoption of multicore CPUs, the number of developers that have to optimize their applications for parallel performance can be expected to increase significantly.

The most common approaches for performance analysis are profiling and tracing, the former often being preferred due to its lower overheads and more easily comprehensible results and the latter finding most use for message passing applications. Most current tools, profiling as well as tracing, deliver their data correlated to the program source code, e.g., percentage of execution time spent in a particular function, number of bytes transferred in a particular MPI call, or cache misses incurred due to a particular program statement.

* This work was partially supported by US DOE SCIDAC grant #DE-FC02-06ER25761 (PERI) and NSF grant #07075433 (SDCI).

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 263–272, 2009.
© Springer-Verlag Berlin Heidelberg 2009

An aspect of program development that has received less attention, mostly due to limited hardware support, is the dimension of data structures. The objective here is to see how performance metrics are related to data structures in addition to source code regions. With the advanced performance monitoring unit of Intel's Itanium processor series such an analysis becomes possible.

This paper describes how the hardware features of the Itanium 2 processor are exploited by the perfmon and PAPI performance monitoring APIs and how PAPI's support for address range restrictions has been integrated into an existing profiling tool to achieve the goal of data structure oriented profiling in the context of shared memory applications.

The rest of this paper is organized as follows: In Sect. 2 we describe the basic hardware capabilities of the Itanium 2 processor and how they are made available through PAPI. We then describe how we integrated the address range restriction capabilities into an existing profiling tool for OpenMP and made them available to the user. In Sect. 3 we demonstrate the application of the extended tool to a simple example application. In Sect. 4 we describe related work in the area and in Sects. 5 and 6 we describe our plans for continuing our work and conclude, respectively.

2 Data Structure Oriented Profiling

This section describes the hardware capabilities for data address range restricted monitoring and how they are made available through the layers of perfmon2 and PAPI in a profiling tool for OpenMP applications. perfmon2 is a generic low-level interface to the Performance Monitoring Unit of modern microprocessors which is currently implemented for Itanium, IA-32, x86-64, and PPC64 architectures [9]. PAPI is a cross-platform interface to the hardware counters supported by the performance monitoring unit (PMU) that includes portable routines as well as a standard set of performance metrics [1]. PAPI is layered on top of perfmon2 on perfmon2-supported architectures.

2.1 Hardware Capabilities and Their Availability through Perfmon and PAPI

Event counting on the Itanium 2 processor can be qualified by a number of conditions, including instruction address, opcode matching, and data address ranges. Of the roughly 475 native events available on the Itanium 2, 160 of them are memory related and can be counted with data address specification in place. In addition to data address restrictions, the Itanium 2 supports restrictions with respect to instruction addresses, which is however not discussed in this paper.

To specify the data address range qualification, four pairs of special registers are available. The starting and ending addresses cannot be specified exactly, since the hardware representation relies on powers-of-two bitmasks. The perfmon library used by PAPI tries to optimize the alignment of these power-of-two regions to cover the addresses requested as effectively as possible with the four

sets of registers available. Perfmon first finds the largest power-of-two address region completely contained within the requested addresses. Then it finds successively smaller power-of-two regions to cover the errors on the high and low end of the requested address range. The effective result is that the actual range specified is always equal to or larger than and completely contains the requested range, and can occupy from one to four pairs of address registers. In some cases this can result in significant overcounts of the events of interest, especially if two active data structures are located in close proximity to each other. This may require that the developer insert some padding structures before and/or after a particular structure of interest to guarantee accurate counts (although the padding may introduce additional perturbations).

The PAPI team has implemented a generalized interface for data structure and instruction range performance instrumentation beginning with the PAPI 3.5 release. Since PAPI is a platform-independent library, care must be taken when extending its feature set so as not to disrupt the existing interface or clutter the API with calls to functionality that is not available on a large subset of the supported platforms. To that end, the PAPI developers elected to extend an existing PAPI call, `PAPI_set_opt()`, with the capability of specifying starting and ending addresses of data structures or instructions to be instrumented. The `PAPI_set_opt()` call previously supported functionality to set a variety of optional capability in the PAPI interface, including debug levels, multiplexing of eventsets, and the scope of counting domains. This call was extended with two new cases to support instruction and data address range specification: `PAPI_INSTR_ADDRESS` and `PAPI_DATA_ADDRESS`. To access these options, a user initializes a simple option specific data structure and calls `PAPI_set_opt()` as illustrated in the code fragment below:

```
...
option.addr.eventset = EventSet;
option.addr.start = (caddr_t)array;
option.addr.end = (caddr_t)(array + size_array);
retval = PAPI_set_opt(PAPI_DATA_ADDRESS, &option);
...
```

The user creates a PAPI eventset and determines the starting and ending addresses of the data to be monitored. The call to `PAPI_set_opt()` then prepares the interface to count events that occur on accesses to data in that range. The specific events to be monitored can be added to the eventset either before or after the data range is specified.

It is important that the user has some way to know what approximations have been made, so that appropriate corrective action can be taken. For instance, to isolate a specific data structure completely, it may be necessary to pad memory before and after the structure with dummy structures that are never accessed. To facilitate this, `PAPI_set_opt()` returns the offsets from the requested starting and ending addresses as they were actually programmed into the hardware. If the addresses were mapped exactly, these values are zero.

2.2 Data Structure Monitoring Support in ompP

ompP is a profiling tool for OpenMP applications designed for Unix-like systems. ompP differs from other profiling tools like gprof [4] or OProfile [7] in primarily two ways. Firstly, ompP is a measurement based profiler and does not use program counter sampling. The instrumented application invokes ompP monitoring routines that enable a direct observation of program execution events (like entering or exiting a critical section). An advantage of the direct approach is that the results give exact counts, rather than sampled values, and hence they can even be used for correctness testing.

The second difference lies in the way of data collection and representation. While general profilers work on the level of functions, ompP collects and displays performance data in the user model of the execution of OpenMP events [5]. For example, the data reported for critical section contains not only the execution time but also lists the time to enter and exit the critical construct (enterT and exitT, respectively) as well as the accumulated time each thread spends inside the critical construct (bodyT) and the number of times each thread enters the construct (execC). An example profile for a critical section is given in Fig. 1.

```
R00002 main.c (20-23) (unnamed) CRITICAL
 TID      execT      execC      bodyT     enterT      exitT     L3_MISSES
   0       1.00          1       1.00       0.00       0.00       534 513
   1       3.01          1       1.00       2.00       0.00       534 733
   2       2.00          1       1.00       1.00       0.00       535 420
   3       4.01          1       1.00       3.01       0.00       535 062
 SUM      10.02          4       4.01       6.01       0.00     2 139 728
```

Fig. 1. Profiling data delivered by ompP for a critical section. execC denotes the execution count, enterT, exitT, bodyT, and execT are timing data in seconds for entering, exiting, executing the body of the critical section. execT is the sum of all other reported times.

Profiling data in a style similar to that shown in Fig. 1 for critical sections are delivered for each OpenMP construct, with the columns (execution times and counts) depending on the particular construct. Furthermore, ompP supports the query of hardware performance counters through PAPI [1] and the measured counter values appear as additional columns in the profiles. In addition to OpenMP constructs that are instrumented automatically using Opari [8], a user can mark arbitrary source code regions such as functions or program phases using a manual instrumentation mechanism. Function calls can be automatically instrumented with compilers that support this feature.

Profiling data are displayed by ompP as flat profiles and as callgraph profiles, giving both inclusive and exclusive times in the latter case. As an advanced productivity feature, ompP performs overhead analysis in which four well-defined overhead classes (synchronization, load imbalance, thread management, limited parallelism) are quantitatively evaluated.

To support data structure oriented profiling, mechanisms that allow the user to specify the address range of data structures as well as changes in the result reporting are required. Several methods for specifying the data structure to be monitored have been implemented in ompP. First, for global and statically allocated data structures, a data structure's symbol name can be supplied when invoking ompP. For static and global data structures the name can be associated with the correct virtual address start and size of the symbol by invoking the nm tool provided that the debug information is contained in the binaries. In addition to this automatic conversion from symbol names to address ranges, the user can manually specify name, data address start and range through environment variables. For example:

```
# export OMPP_DATA_ADDR=0x6000000000022f80
# export OMPP_DATA_SIZE=34000
# export OMPP_DATA_NAME=tilearray
# ./ompp myprogram
```

For dynamically allocated memory or stack variables, the above mechanisms are neither adequate nor convenient. For this reason, ompP also provides a programmatic way to select the address range, size and name to monitor. A developer can annotate the source code with direct calls to set up the monitoring. For example:

```
double vec[1234];
...
ompp_papi_set_drange(vec, sizeof(vec), "vec")
```

will set up monitoring for the vec array. Alternatively it is possible to use source code annotations in the form of pragmas (in C/C++) or special comments (FORTRAN) that are translated to ompp_papi_set_drange() calls by the Opari preprocessor, for example:

```
#pragma pomp inst data(vec, sizeof(vec), "vec")
```

The latter technique has the advantage that has no compile- or link time dependency on ompP remains because the annotations can just be ignored by compilers.

The data reporting side of ompP has been changed to account for the additional data available through the address range restriction. The header section of ompP's profiling report lists the address range restrictions that are in effect and the offsets that occur due to imprecise coverage of the range with Itanium 2's four coarse mode counters as shown below. This information is important when interpreting the profiling reports to exclude the possibility to falsely attribute data to neighboring data structures.

```
Data Address    : 0x6000000000022f80
Data Size       : 34000 (0x84D0)
Data Name       : tilearray
Data Start Offs.: 0
Data End   Offs.: 0
```

For the bookkeeping of profiling data, ompP introduces execution overhead in the measured application that is directly proportional to the number of region enter or exit operations monitored. For reasonably "well behaved" applications, such as the SPEC OpenMP benchmarks, `ompP`'s monitoring overhead usually is below five percent of execution time with PAPI enabled. We could not observe additional overhead using Itanium's data address range restriction feature over using the counters without this feature enabled.

3 Experimental Evaluation

In this section we show a simple application example of the data structure monitoring capabilities of `ompP`. The program fragment shown in Fig. 2 implements a very simple matrix $\times$ vector multiplication $Ax = b$ and the purpose is to show how the memory system related events can be restricted to only those occurring for the matrix A as an example. Consider this program fragment in Fig. 2 and note that data range monitoring is set up for the `a` array with the proper size.

```
  integer n, i, j
  parameter( n=12000 )

  double precision a(n,n)
  double precision b(n)
  double precision c(n)
...
!$POMP INST DATA( a , n*n*8 , "a" )

!$OMP PARALLEL DO private(i,j)
  DO i = 1, n
    c(i) = 0.0;
    DO j = 1, n
      c(i)=c(i)+a(i,j)*b(j)
    END DO
  END DO
...
```

Fig. 2. Matrix $\times$ vector multiplication example for data address range restrictions

The execution of this application on a 4-way Itanium 2 ("Madison Processor") SMP machine with 1.3 GHz, 3 MB third level cache and 8 GB of main memory resulted in a profiling report that shows the address range specification in place:

```
Data Address     : 0x6000000000026f40
Data Size        : 1152000000  (0x6DDD000)
Data Name        : a
Data Start Offs.: 159552  (0x26f40)
Data End   Offs.: 55800000  (0x35370c0)
```

Note the extra covering of the array due to imprecise mode. After padding the array at the beginning and at the end (with a similar sized pad1 and pad2 array, in this case) and setting counters to measure load instructions (PAPI_LD_INS) we get the profile shown below for the main parallel loop. Note that these numbers are exactly what one would expect to see: Each of the four threads works on a sub-matrix of size $3\,000 \times 12\,000 = 36\,000\,000$ and has to bring this into registers. Also note that the loads for the other data structures are not included.

```
R00002 main.f (30-37) LOOP
 TID       execT       execC       bodyT   exitBarT     PAPI_LD_INS
   0        4.66           1        4.66       0.00      36 000 000
   1        5.24           1        5.24       0.00      36 000 000
   2        5.23           1        5.23       0.00      36 000 000
   3        4.91           1        4.91       0.00      36 000 000
 SUM       20.04           4       20.04       0.00     144 000 000
```

Next we analyzed the memory system performance of the code by measuring the number of level three misses, as shown in the following profile:

```
R00002 main.f (30-37) LOOP
 TID       execT       execC       bodyT   exitBarT       L3_MISSES
   0        5.01           1        5.01       0.00      33 735 611
   1        5.24           1        5.24       0.00      35 447 540
   2        5.22           1        5.22       0.00      35 470 957
   3        5.14           1        5.14       0.00      35 004 338
 SUM       20.61           4       20.61       0.00     139 658 446
```

Evidently, a very high percentage of the loads fail the L3 cache and have to go to memory. Looking at the source code of the multiplication operation the reason is obvious. The array is accessed with a stride of the matrix dimension and not linearly, leading to very poor cache reuse. Changing the two indexes of the matrix multiplication we arrive at a version with much better cache performance: Also note the improved execution time (0.64 seconds vs. 5.20 seconds).

```
R00002 main.f (30-37) LOOP
 TID       execT       execC       bodyT   exitBarT       L3_MISSES
   0        0.64           1        0.64       0.00       2 250 582
   1        0.64           1        0.64       0.00       2 250 741
   2        0.64           1        0.64       0.00       2 250 734
   3        0.64           1        0.64       0.00       2 250 730
 SUM        2.54           4        2.54       0.00       9 002 787
```

While the shown example is trivial, it demonstrates the main benefit of the data address range restriction: contributions of individual elements can be singled out from the overall summed hardware counter values measured and hence another dimension in which performance analysis can be conducted (that along different data structures) opens up. If, for example, the same matrix `a` was accessed in several routines, or parallel loops, a restriction to `a` would show if it

is accessed with similar efficiency in all routines. A wrong indexing in one place would stick out in comparison to the other cases, something that might go unnoticed in the wealth of counter data produced by the overall routine, if the restriction to the array was not in place.

4 Related Work

The value of analyzing programs with a focus on data structures has been recognized before. There are proposals [6,2] for new or extended hardware monitoring units that allow a more detailed analysis of cache behavior of data structures such as uncovering the reason for eviction of cache lines.

With respect to actual hardware, the Itanium 2 processor is the only microprocessor available today that allows data structure oriented profiling to be used in practice. Although the `dprof` tool [10,3] exploits these capabilities with goals similar to ours, the techniques used are different. `dprof` uses the Itanium's EARs (event address registers) that can sample load operations, their code and data addresses and latency. This allows basically all data structures to be monitored simultaneously. The association between virtual addresses and data structures is accomplished by using the executable's debug information and by capturing `malloc()` calls. In comparison, our technique can only monitor one data structure at a time, but the results obtained are exact and not subject to sampling inaccuracy. Also, the EARs only monitor load operations and the association to which level of the memory hierarchy satisfied the load has to be done by analyzing the latency, while our technique allows us to monitor all events related to the memory subsystem.

5 Future Work

We plan to explore the data address oriented profiling approach along several directions in the future. First, to overcome the limitation of being able to monitor only one address range at a time, we plan to combine the current direct measurement technique with sampling in the following way: The developer will be enabled to specify several data structures to be monitored and those will be kept in a list. In addition, a PAPI overflow event can be set up (either based on the elapsed CPU cycles or on the memory-system related event to be monitored) and on each overflow the monitoring switches to another data structure. A comparison with the EAR-based approach as realized in `dprof` will show which of the two approaches delivers better results in terms of accuracy and ease of use.

A second direction in which a more detailed investigation is warranted is the area of the interaction of threads and data structure. In the current implementation, the same address range restriction is set up for each thread in `ompP`, while a separate event set (and range) is supported by PAPI. With globally allocated data structures shared across threads this is not a problem, but separate data structures per thread are needed to fully support analysis of privatized arrays in OpenMP.

We plan to validate our data-structured oriented approach to performance analysis by using it with applications having known problems with data access. This investigation will compare the ease of our approach with the labor-intensive manual instrumentation, modeling, and analysis that has been used to find and fix these problems. Following this comparison, we plan to use our approach to detect similar problems in large-scale applications.

6 Conclusion

We have discussed the hardware capabilities of the Intel Itanium 2 processor for data address range restrictions and how those capabilities, which are available through the PAPI interface, are used in a profiling tool for OpenMP applications for data structure oriented profiling. From the user's perspective, a central issue is the ability to specify the data items to be measured which we have addressed by offering code markup methods as well as runtime specification through environment variables. An appealing property is the exactness of the delivered results – for load and store operations, for example, the measured data usually fit exactly the expected results. Also, all memory system related events (160) are available for measurement.

There are also some shortcomings with the data address range restrictions as presented here. The most severe issue is the inexact coverage of the data address range due to hardware restrictions which leads to offsets at the beginning and end of the covered range. Padding is needed to move the monitored data structure into an area surrounded by inactive regions of memory that will not disturb the measurement. Naturally, such a padding could have adverse effects on the application's performance and defeat the purpose of discovering if and which padding is required for optimization purposes. To circumvent this issue we plan to investigate a mode where the range is not selected to be minimally including, but maximally fitting the requested range. That would solve problems of wrong attribution at the expense of less accurate results.

References

1. Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.J.: A portable programming interface for performance evaluation on modern processors. Int. J. High Perform. Comput. Appl. 14(3), 189–204 (2000)
2. Buck, B.R., Hollingsworth, J.K.: A new hardware monitor design to measure data structure-specific cache eviction information. International Journal of High Performance Computing Applications 20(3), 353–363 (2006)
3. Gaugler, T.: Ein Werkzeug zur Untersuchung des Cacheverhaltens von Datenstrukturen mittels Ereigniszählern. Diplomarbeit, Universität Karlsruhe (2005)
4. Graham, S.L., Kessler, P.B., McKusick, M.K.: Gprof: A call graph execution profiler. SIGPLAN Not. 17(6), 120–126 (1982)
5. Itzkowitz, M., Mazurov, O., Copty, N., Lin, Y.: An OpenMP runtime API for profiling. The OpenMP ARB as an official ARB White Paper (accepted), http://www.compunity.org/futures/omp-api.html

6. Kereku, E., Li, T., Gerndt, M., Weidendorfer, J.: A data structure oriented monitoring environment for fortran openMP programs. In: Danelutto, M., Vanneschi, M., Laforenza, D. (eds.) Euro-Par 2004. LNCS, vol. 3149, pp. 133–140. Springer, Heidelberg (2004)
7. Levon, J.: OProfile, A system-wide profiler for Linux systems, `http://oprofile.sourceforge.net`
8. Mohr, B., Malony, A.D., Shende, S.S., Wolf, F.: Towards a performance tool interface for OpenMP: An approach based on directive rewriting. In: Proceedings of the Third Workshop on OpenMP, EWOMP 2001 (September 2001)
9. Perfmon2, the hardware-based performance monitoring interface for linux, `http://perfmon2.sourceforge.net/`
10. Tao, J., Gaugler, T., Karl, W.: A profiling tool for detecting cachecritical data structures. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 52–61. Springer, Heidelberg (2007)

Complete Def-Use Analysis in Recursive Programs with Dynamic Data Structures*

R. Castillo, F. Corbera, A. Navarro, R. Asenjo, and E.L. Zapata

Dpt. of Computer Architecture, University of Málaga,
Complejo Tecnologico, Campus de Teatinos, E-29071. Málaga, Spain
{rosa,corbera,angeles,asenjo,ezapata}@ac.uma.es

Abstract. Based on some of our previous works, we show in this paper the possibility of accelerating complex analyses (such as shape analyses, dependence analyses...) thanks to a complete def-use chain analysis. In particular, we put our efforts in accelerating the shape analysis technique developed in our research group. Using the gathered Def-Use(DU) information, we have implemented a code slicing pass that computes the relevant statements required by a client analysis. This work is part of a heap-directed pointer analysis framework, where our final goal is the automatic parallelization of codes based on heap-stored dynamic/recursive data structures.

1 Introduction

Fig. 1 shows the general framework where our heap-directed pointer analysis makes contribution. Our final goal is the automatic parallelization of sequential programs. For that purpose, we need to know if the program exhibits dependences or not. When dealing with codes based on dynamic data structures, some pointer analysis techniques fail to find hidden parallelism. An efficient data dependence test is a required step in the detection of parallelism. With these kind of codes, the data dependence test needs information about the properties of the data structures traversed in the loops or in function bodies. This leads us to shape analysis techniques which are able to capture, at compile time, the shape of dynamically allocated data structures, i.e., those allocated at runtime and accessed through heap-directed pointers. By definition, a shape analysis is a static technique that achieves abstraction of dynamic memory, so it can help to disambiguate, quite accurately, memory references in programs that create and traverse recursive data structures. Our current dependence test is based in the *Shape Analysis Tool* [1] that executes symbolically all code statements. The shape analysis it performs is very costly, so a preprocessing pass was designed to improve this situation. A slicing pass was added, based on the information provided by an *Interprocedural DefUse*(DU) chain analysis. The aim of the slicing is to reduce the number of statements to analyze, taking only the ones that

* This work was supported in part by the Ministry of Education of Spain under contract TIN2006-01078.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 273–282, 2009.
© Springer-Verlag Berlin Heidelberg 2009

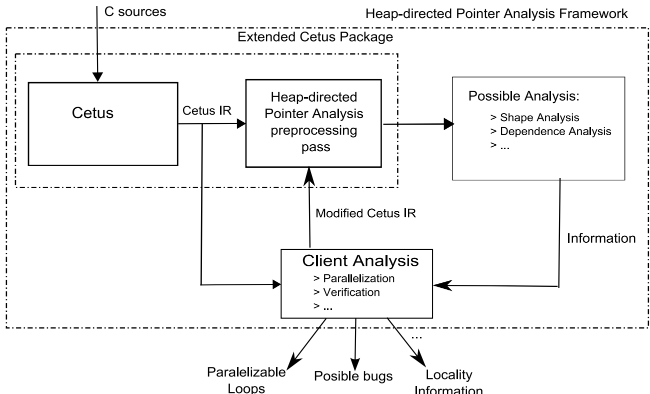


Fig. 1. General heap-directed pointer analysis framework

define the shape of the data structure. Thus, the shape analysis only executes symbolically those statements, and not all statements of the program. As a result, the analysis time for the shape analysis is significantly reduced. We present in this paper our slicing algorithm and show experimental evidence on how it can accelerate a client shape analysis. This shape analysis is crucial for our data dependence detection test as we have mentioned before.

Part of the DU Interprocedural analysis is based in the technique of Hwang [2], which specializes in the identification of interprocedural DU chains of dynamic recursive data structures. It is an iterative data flow analysis that first gathers local DU and alias information inside each procedure, then propagates this local information through a special interprocedural graph, and finally computes the DU chains in the program with an extra intraprocedural step over each procedure. Thanks to this technique, not only intraprocedural DU chains are detected, but interprocedural ones as well. The rest of DU chains due to pointers (not accessing recursive fields) is computed through so-called *Interprocedural Static Single Assignment*, ISSA. We put all this information to work in a slicing pass. First, an initial group of *seed* statements (relevant to the shape analysis criteria) are selected. These statements mark points of interest in the code for the technique. The slicing pass is operated by a worklist algorithm that ends when there are no more statements to analyze. The result of this slicing pass is a subset of the statements in the program involved in the data structure creation. Basically, we filter out other statements related to the structure traversal which in fact do not contribute to the shape of the structure. Our goal is to produce, with just the relevant creation statements, the same shape analysis result which is obtained with the original code. As a consequence, we obtain a good acceleration of the shape analysis (see the experimental results section), taking less time to find the same result.

All our algorithms have been implemented in Java and integrated into the Cetus compiler infrastructure [3], adding several new extra passes to this tool.

We use Cetus parsing abilities to obtain an intermediate representation of the code to analyze, where we can perform our passes.

The rest of this paper is organized as follows: we describe details of the basic DU chain analysis in Section 2, and the slicing pass in Section 3. Experimental results are introduced in Section 4. In Section 5 some related works can be found, and finally we talk about conclusions and future work in Section 6.

2 DU Chain Analysis

DU information can be used to identify statements directly involved in the creation of recursive data structures that are referenced in the segment of code under analysis. A DU chain establishes a relationship between the definition statement where a link is created and each statement where it is used. With that information we can automatically detect which statements actually define the shape of dynamic memory and discard all other statements in a program. For this purpose, we have implemented two algorithms: a *interprocedural link DU* technique, based on Hwang's work, and a *interprocedural pointer DU* technique, based on ISSA representation. The interprocedural link DU technique, described in Fig. 2, can be divided in four stages (four boxes in the figure). There is an initial intraprocedural analysis on each local procedure of the program, where DU tuples are created for each pointer accessing a recursive field, such as `p->f`. Once the fixed point is reached, all local information of pointer accessing fields is recorded, as a tuple, in a special graph called *Interprocedural Flow Graph* (IFG), made of nodes and links. At the same time, information about how formal and actual parameters are transformed inside each procedure is collected, and used to build the corresponding IFG edges. The complete IFG of the program is built in the stage two. There are four types of nodes (ENTRY, EXIT, CALL and RETURN) and three different types of edges (Reaching, Binding and Interreaching).

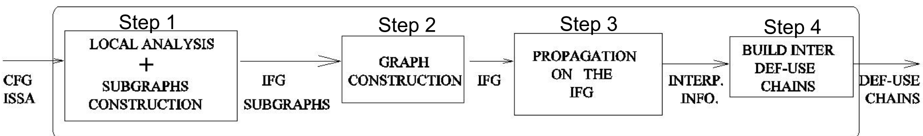


Fig. 2. Steps in the interprocedural link algorithm

In the following step, stage three of Fig. 2, tuples are propagated and transformed along the whole IFG. This propagation step simulates the possible flow of information that may occur between different procedures due to function call sites. Contexts in function calls are preserved, so only realizable paths are traversed. For this reason, propagation is split in two phases: in the first phase of propagation, information goes from the caller procedures to callee procedures (from an outer level to an inner one); in a second phase information goes in the

```
int main (void){
    node *a, *b, *c;
    a=malloc();
    b=malloc();
    c=f(a,b);
    g(a,b);

}
node f (node *x,node *y){
    x->nxt=y;
    x=y;
    return x;
}
void g (node *s,node *t){
    node *v;
    v=f(s,t);
}
```

(a)

```
int main (void){
S1: a_0_1=malloc();
S2: b_0_1=malloc();
S3: c_0_1=f(a_0_1, b_0_1);
S4: c_0_1=x_1_0;
S5: a_1_1=x_1_0;
S6: b_1_1=y_0_0;
S7: g(a_1_1,b_1_1);
S8: a_2_1=s_1_2;
S9: b_2_1=t_1_2;
}
node f (node *x, node *y){
S10:    x_0_0=Iphi(a_0_1, s_0_2);
S11:    y_0_0=Iphi(b_0_1, t_0_2);
S12:    x_0_0->nxt=y_0_0;
S13:    x_1_0=y_0_0;
S14:    return x_1_0;
}
void g (node *s, node *t){
S15:    s_0_2=Iphi(a_1_1);
S16:    t_0_2=Iphi(b_1_1);
S17:    v_0_1=f(s_0_2, t_0_2);
S18:    v_0_1=x_1_0;
S19:    s_1_2=x_1_0;
S20:    t_1_2=y_0_0;
```

(b)

Fig. 3. (a) Code of a simple program;(b) ISSA-form of the code

other way, from callee procedures to caller procedures (from an inner level to the outer one). When the fixed point is reached, DU information is passed from the IFG to local Flow Graphs of procedures. Finally, a last intraprocedural pass is done for the detection of interprocedural DU chains.

This is the general idea of how this technique works, however there are some initial conditions to take into account for the correct execution of this analysis.

- a Flow Graph of the program is needed, so we have chosen the Control Flow Graph (CFG) of the program provided by Cetus and we have extended it with some useful properties, such as the depth level of each block in the CFG.
- each variable of the code must have a unique and global identity in the whole program. We have implemented the *Interprocedural Static Single Assignment*(ISSA) form (Fig. 3) of the program. With these representation we can guarantee that each variable has a unique and non-repeated identifier in the whole program. We will explain more details about how it works later.

We have adapted both our platform of work, Cetus, and our codes to match these pre-requisites. The accommodation of Cetus is possible because it is an extensible infrastructure, in that way we have added several new passes to it.

The ISSA pass can be considered as an extension of the original SSA form. The original SSA form ensures that each variable has a unique identity inside each procedure, not in the whole program, so there can be two variables with the same name in different procedures. The SSA pass was already added to Cetus in one of our previous works. We found out some ideas about how to extend traditional SSA form to fulfill the requirements of the interprocedural technique [4]. Based in these ideas, we added new special statements into some points of the program: a)after each function call and b)at the beginning of each procedure body that is called. The purpose of the first kind of statements is the matching between actual and formal parameters when returning from the procedure. Within the formal parameters, we need the last identifier used for them. The second kind of statements have an inverse functionality, they summarize in one sentence all actual parameters (one for each call to the procedure from different call-sites in the program) connected with the same formal parameter. Thus, they look like phi-statements in SSA form, so we called them Interprocedural phi, *Iphi*, statements. In Fig. 3(a) and Fig. 3(b), we have a simple code written in C and its ISSA-form. Sentences S4, S5, S6, S8, S9, S18, S19 and S20 are of type a). Sentences S10, S11, S15 and S16 are of the second type, *Iphi* statements. Thanks to the ISSA form, we developed other algorithm to capture DU chains not taken into account by the interprocedural link DU technique described before.

```
void f(node *x, node *y){
    node *z, *t;
    z=(node*)malloc();
    x->nxt=z;
    t=y->nxt;
}
void main(){
    node *p, *r, *q;
    p=(node*)malloc();
    q=(node*)malloc();
    p->nxt=q;
    r=p->nxt;
    f(r,q);
}
```

(a)

```
Detected by Interp. Link Algorithm:
    p->nxt=q; <->  r=p->nxt;
    x->nxt=z; <->  t=y->nxt;
Detected by ISSA form:
    q=(node*)malloc()); <-> p->nxt=q;
    z=(node*)malloc()); <-> x->nxt=z;
```

(b)

Fig. 4. (a) Code of a simple program;(b) Different DU chains detected

As an output of the complete DU chain analysis, we have two lists: the DU chains detected by the interprocedural link DU technique, and the DU chains detected by the ISSA form. In Fig. 4(a) and Fig. 4(b), we show a simple code written in C, and some DU chains that would be detected by both processes. There is an intraprocedural DU chain detected in `main` by the interprocedural link technique, and another DU chain in procedure `f` that is detected thanks to the ISSA technique.

3 Slicing Process

Slicing is a technique that extracts statements, relevant to a particular criteria, from a program. It has been applied to many fields, such as debugging, testing, program comprehension, restructuring, downsizing and parallelization [5]. There are also experiments about the good performance of program slicing in recursive programs with dynamic pointer information [6].

```
void main(){
   root=TreeAlloc(level);
   TreeAdd(root);
   return(0);
}
tree *TreeAlloc(int level){
 if (level==0){
    new=NULL;
 }else{
    new_level=level-1;
    new=(struct tree *)malloc();
    new->val=1;
    left=TreeAlloc(new_level);
    new->left=left;
    left=NULL;
    right=TreeAlloc(new_level);
    new->right=right;
    right=NULL;
 }
 return new;
}
```

```
int TreeAdd(tree *t){
 if (t==NULL){
    total_val=0;
  }else{
    tleft=t->left;
    leftval=TreeAdd(tleft);
    tleft=NULL;
    tright=t->right;
    rightval=TreeAdd(tright);
    tright=NULL;
    value=t->val;
    total_val=value+leftval
                  +rightval;
    t->val=total_val;
  }
  return total_val;
}
```

Fig. 5. Code from Treeadd program

We use, as *Seed* statements for this slicing process, the heap traversing statements of the structure. They allow us to isolate the statements related to the creation and connection of heap elements. For instance, in Fig. 5 we show the TreeAdd code from the olden suite [7]. Since we are interested in the parallelization of the TreeAdd procedure body, particularly our dependence test would check if the two TreeAdd calls inside TreeAdd procedure are independent. In this example, seed statements would be `tleft=t->left` and `tright=t->right`. First, this group of selected statements is introduced in a worklist. During the execution of the slicing algorithm, more statements can be added to the worklist. While there are statements to analyze in the worklist, the process continues. Beginning with the uses in the statement, the algorithm finds definitions for those uses. Once definitions are reached and there are no more statements to analyze, the algorithm finishes. Finally, we have an output list with all statements of interest that contain information about where data structures are created. Fig. 6 shows the Treeadd program from Fig. 5 after applying the slicing algorithm.

```
void main(){
   ;
   return(0);
}
tree *TreeAlloc(int level){
 if (level==0){
    ;
 }else{
    new=(struct tree *)malloc();
    ;
    left=TreeAlloc(new_level);
    new->left=left;
    ;
    right=TreeAlloc(new_level);
    new->right=right;
    ;
 }
 return new;
}
```

```
int TreeAdd(tree *t){
 if (t==NULL){
    ;
  }else{
    ;
    ;
    ;
    ;
    ;
    ;
    ;
  }
  return total_val;
}
```

Fig. 6. Sliced code from treeadd program. This code will be passed to the shape analyzer

With the programs we have checked, this slicing pass has been the key to reduce the analysis times of the shape analysis. We believe this slicing process can be of great value to enable our shape analysis technique to analyze more realistic benchmarks. The reason for this result is because the slicing process only gets statements that modify the shape of data structures, and do not get the ones traversing the structure. Usually, the number of statements modifying the shape of data structures is small.

4 Experimental Results

For these tests we have considered some programs that are representative of typical recursive data structures. Two codes have been taken from the olden suite [7] and we also have two programs that make operations with sparse matrixes (one computing the product of a sparse matrix by a sparse vector, and the other the product of both sparse matrixes). Execution time has been measured in an Intel(R) Core^{TM} 2 CPU1.86 GHz with 2 GB RAM.

Fig. 7 displays some metrics for the complete DU-chain analysis performed. The purpose of this graphic is to help us detect the most expensive parts in the analysis, in order to try to optimize them. Below, next to the names of the codes, we can see the total times measured for the whole analysis. We have divided the analysis times in five parts. Four of them (*intra, ifg, propag, inter* which corresponds to the ones shown in Fig. 2) belong to the interprocedural link algorithm, and a last part with the computation of ISSA DU chains and the slicing pass (*lastF*). The graphic shows the percentage of the total time

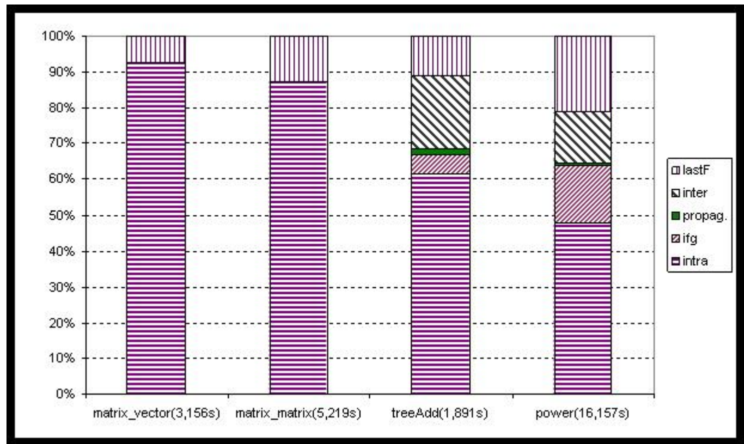


Fig. 7. Measures for different stages in the DU chain analysis

Table 1. Analysis time measured for the shape analysis tool

Program	Original #stmts	Filtered #stmts	Original Time	Filt. Time	%time reduction
Mat-Vec	155	57	2.947s	0.531s	81.98%
Mat-Mat	200	80	27.1s	1.3s	95.20%
TreeAdd	42	17	5.518s	0.591s	89.28%
Power	272	60	4.919s	1.839s	62.61%

taken for every separate part of the analysis. The intraprocedural analysis part collects all information required by the following stages, and turns out to be the most expensive. Watching the graphic, we see that the computation of ISSA DU chains and the slicing part are less than 20% of the total time in most cases. In codes TreeAdd and power, stages *ifg* and *propa* are a negligible percentage of the total time of the analysis.

Then in Table 1, we present some experimental evidence regarding the gathered acceleration obtained by the shape analyzer, when comparing the analysis times before and after the code slicing. First column shows the original size of the program in number of statements. The following column shows the size after the slicing pass. The third column contains the execution times in seconds of the shape analysis over the original program. The fourth column shows the times in seconds for the same shape analysis but over the pruned program. In all cases, sliced programs produce the same output graphs than their original counterparts, capturing memory configuration without any loss in precision. In addition, a time reduction of more than 60% is achieved, up to a 95,2% reduction in the best case for the multiplication of matrixes program. These results prove the effectiveness of our approach.

5 Related Work

There are some initial works by Wolfe et al. that used SSA in parallelism [8] but they do not deal with pointers. Then, Hendren et al. [9] proposed an interesting SSA application for pointers. However, their extended SSA numbering has no phi-nodes, so it cannot be used to get DU chains.

Harrold and Soffa's work on interprocedural DU chains [10] inspired the algorithm developed by Hwang and Saltz [4]. Part of the work presented in this paper is based in this last approach. They also proposed a shape analysis application for the interprocedural algorithm, quite similar to Ghiya and Hendren's one [11]. Our shape analysis works in a different way, inspired by Sagiv et al.[12]. It gives a representative graph with all collected information about heap locations in the program.

Regarding other interprocedural approaches, Assman and Weinhard [13] proposed a worklist algorithm using summary nodes and so-called storage shape graphs with skeleton trees of definitions and uses of nodes. They used procedure cloning which is quite expensive in large programs or with a high level of recursion. Another interprocedural work, by Guyer and Lin [14], uses a worklist algorithm based in the computation of DU chains. Their technique acknowledges some performance reduction with programs based in recursive data structures.

6 Conclusions and Future Work

In this paper, we have shown one of the possible applications for the complete DU chain analysis implemented. We demonstrated how a shape analysis technique can be accelerated thanks to a code slicing pass based on the precise information applied by the DU chain technique. The DU analysis can be extended with the automatic detection in the code of so-called *induction pointers*. They are used in loops to traverse recursive structures, establishing the traversal pattern. That pattern, along with the shape of the data structure, allows to detect dependencies between accesses. Of course, induction pointers already introduce an inherent dependence between different iterations of a loop, something known as the pointer-chasing problem. However, there are techniques to overcome it, provided that no other dependencies exist [15]. Being able to automatically detect induction pointers is the next pass for our compiler analysis framework, because they are needed to identify loops that traverse recursive data structures. These loops are candidate for parallelization in our approach. It is also important for the dependence analysis to detect access paths in the program, so we also plan to extend the current analysis for their automatic detection.

Besides, we are currently working in the implementation of a new dependence test analysis based in the work presented in this paper. Some preliminary experimental results show that a significant time reduction can be achieved comparing executions between our current dependence analysis [16] and the new proposed one. However all this recent work will be included in a future paper.

References

1. Asenjo, R., Castillo, R., Corbera, F., Navarro, A., Tineo, A., Zapata, E.L.: Parallelizing irregular C codes assisted by interprocedural shape analysis. In: 22nd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2008), Florida, USA (April 2008)
2. Hwang, Y.S.: Interprocedural definition-use chains of dynamic recursive data structures. Ph.D. thesis, Faculty of the Graduate School of the Univ. of Maryland (1998)
3. Johnson, T.A., Lee, S.I., Fei, L., Basumallik, A., Upadhyaya, G., Eigenmann, R., Midkiff, S.P.: Experiences in using Cetus for source-to-source transformations. In: Eigenmann, R., Li, Z., Midkiff, S.P. (eds.) LCPC 2004. LNCS, vol. 3602, pp. 1–14. Springer, Heidelberg (2005)
4. Hwang, Y., Saltz, J.: Identifying DEF/USE information of statements that construct and traverse dynamic recursive data structures. In: Huang, C.-H., Sadayappan, P., Sehr, D. (eds.) LCPC 1997. LNCS, vol. 1366, pp. 131–145. Springer, Heidelberg (1998)
5. Lucia, A.D.: Program slicing: Methods and applications. In: Proc. 1st IEEE International Workshop Source Code Analysis and Manipulation (2001)
6. Mock, M., Atkinson, D.C., Chambers, C., Eggers, S.J.: Program slicing with dynamic points-to sets. IEEE Transactions on Software Engineering 31(8), 657–678 (2005)
7. Carlisle, M., Rogers, A.: Software caching and computation migration in olden. In: ACM Symposium on Principles and Practice of Parallel Programming, PPoPP (1995)
8. Stoltz, E., Gerlek, M.P., Wolfe, M.: Extended ssa with factored use-def chains to support optimizationand parallelism. Software Technology, Proc. of the 27th Hawaii International Conference on System Sciences 2, 43–52 (1994)
9. Lapkowski, C., Hendren, L.: Extended ssa numbering: Introducing ssa properties to languages with multi-level pointers. In: ACAPS Tech. Memo. Sch. of Comp. Sci., McGill U, vol. 102, pp. 128–143. Springer, Heidelberg (1998)
10. Harrold, M., Soffa, M.: Efficient computation of interprocedural definition-use chains. ACM Trans. Program. Lang. Syst. 16, 175–204 (1994)
11. Ghiya, R., Hendren, L.J.: Is it a tree, a DAG, or a cyclic graph? A shape analysis for heap-directed pointers in C. In: Conference Record of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, St. Petersburg, Florida (1996)
12. Sagiv, M., Reps, T., Wilhelm, R.: Solving shape-analysis problems in languages with destructive updating. ACM Transactions on Programming Languages and Systems 20(1), 1–50 (1998)
13. Assman, U., Weinhardt, M.: Interprocedural heap analysis for parallelizing imperative programs. In: Proc. Programming Models for Massively Parallel Computers, pp. 74–82 (1993)
14. Guyer, S.Z., Lin, C.: Client-driven pointer analysis. In: Cousot, R. (ed.) SAS 2003. LNCS, vol. 2694. Springer, Heidelberg (2003)
15. Ghiya, R., Hendren, L.: Putting pointer analysis to work. In: Proc. 25th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California, pp. 121–133 (January 1998)
16. Navarro, A., Corbera, F., Tineo, A., Asenjo, R., Zapata, E.L.: Detecting loop-carried depedences in programs with dynamic data structures. Journal of Parallel and Distributed Computing 67, 47–62 (2007)

Parametric Studies in Eclipse with TAU and PerfExplorer

Kevin A. Huck, Wyatt Spear, Allen D. Malony, Sameer Shende, and Alan Morris

Performance Research Laboratory
Department of Computer and Information Science
University of Oregon, Eugene, OR, USA
{khuck,spear,malony,shende,amorris}@cs.uoregon.edu
http://www.cs.uoregon.edu/research/tau

Abstract. With support for C/C++, Fortran, MPI, OpenMP, and performance tools, the Eclipse integrated development environment (IDE) is a serious contender as a programming environment for parallel applications. There is interest in adding capabilities in Eclipse for conducting workflows where an application is executed under different scenarios and its outputs are processed. For instance, parametric studies are a requirement in many benchmarking and performance tuning efforts, yet there was no experiment management support available for the Eclipse IDE. In this paper, we describe an extension of the Parallel Tools Platform (PTP) plugin for the Eclipse IDE. The extension provides a graphical user interface for selecting experiment parameters, launches build and run jobs, manages the performance data, and launches an analysis application to process the data. We describe our implementation, and discuss three experiment examples which demonstrate the experiment management support.

Keywords: parallel, performance, experiment management, Eclipse.

1 Introduction

Integrated development environments (IDEs) help to facilitate software development and maintenance. IDEs provide a consistent development environment, numerous enhancements to the development process, and are the standard in industrial software development. IDEs are not very common in parallel application development, but improving toolkit functionality makes it possible to write, compile, launch and run large-scale parallel applications on a local machine or on a remote resource.

Parametric studies are necessary in many benchmarking and performance tuning efforts. This is especially true for parallel applications, where scaling studies are key to exploiting highly parallel hardware for maximum return. Parametric studies are helpful in finding scaling bottlenecks and communication design flaws, and improving algorithmic efficiency. However, parametric studies can

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 283–294, 2009.
© Springer-Verlag Berlin Heidelberg 2009

consist of hundreds or thousands of application configurations, and automated parametric studies can be complex to perform. Traditional parametric studies on parallel hardware requires scripts for building configurations, scripts for submitting batch jobs to the queue, scripts for data management, and the eventual analysis processing at the end of the executions. Script programming is error prone, and particularly costly if mistakes are not found until after hundreds of processing hours have been consumed. Parametric study scripts are frequently re-usable only with considerable effort, as the differences between two or more parallel applications can be significant. There is a clear opportunity to improve upon the parametric study process.

Eclipse [1] is a user configurable software development IDE with a plugin-centric design. Plugins have been developed for a wide range of development purposes. A TAU [2] performance analysis toolkit plugin for Eclipse has been written, and can be used for instrumentation and measurement of C, C++, Fortran and Java applications developed in Eclipse [3,4]. However, there was no mechanism in the various plugins for experiment management with regards to performance studies. For that reason, we extended the Parallel Tools Platform (PTP) plugin for Eclipse to include a parametric study framework for the TAU plugin in Eclipse. Users can set up a desired study, launch the experiment, and the framework will automatically compile and execute the application with the specified configuration combinations, storing the performance result after each run in a PerfDMF [5] repository. When the experiment is complete, the multi-experiment analysis and data mining tool PerfExplorer [6] is launched, and the automated comparative analysis results are produced. While TAU, PerfDMF and PerfExplorer are the tools we used in our experiments, consideration was given to the implementation to ensure that other performance tools could be used in the same framework.

The remainder of this paper is as follows. Section 2 will provide some background discussion of Eclipse and the plugin components involved. Section 3 will describe the experiment management support implementation. Section 4 will describe some analysis examples using the experiment management support. Section 5 will describe related work in parametric study support, and Section 6 will describe our conclusions and future work.

2 Background

Eclipse. Eclipse[1] is a popular software platform with support for customized IDE functionality. Its default set of plugins is designed for Java development, but the Eclipse community has provided support for other languages such as C/C++ and Fortran. Support for high performance computing has also been provided via the Parallel Tools Platform (PTP). Two distinct advantages of the Eclipse platform are its portability and extensibility. The former is provided largely by Eclipse's Java-based implementation, which means it can be run consistently on Windows, Macintosh and many Unix based OSes. Because Eclipse is open source, users are free to modify and extend its functionality as they see fit. As a

result, there is a diverse array of enhancements and plugins available to increase Eclipse's functionality for software development, as well as other tasks. A longer description of Eclipse can be found in [3,4] and elsewhere.

Eclipse Plugins. There are four Eclipse plugin collections, or projects, that are related directly to the integration of the TAU performance analysis tools. The Java Development Tools (JDT) [7], the C/C++ Development Tools (CDT) [8], the Photran Development Environment [9], and the Parallel Tools Platform (PTP) [10] all facilitate the development of programs and the use of programming paradigms that are supported by TAU and none include their own internal mechanisms for performance analysis.

JDT. The JDT assists with Java development by providing a context sensitive source editor, project management and development control facilities, among other features.

CDT. Many of the CDT's features are comparable to those of the JDT. However, the build system of the CDT is naturally quite different. It supports both the use of external makefiles and an internally constructed "Managed" makefile system. In either case the compilation and linking of programs within the CDT is accomplished via user specified compilers and compiler options.

Photran. Photran is a Fortran development environment, based on CDT. Photran has support for Fortran 77, 90 and 95.

PTP. PTP provides the ability to write, compile, launch and debug parallel programs from within Eclipse. PTP supports both OpenMP and MPI based parallelism, and is also based on CDT.

TAU. TAU [2] is a mature performance analysis system designed to operate on many different platforms, operating systems and programming languages. In addition to collecting a wide range of performance data it includes resources for performance data analysis and conversion to third party data formats.

Many of TAU's functions are closely bound to the underlying architecture of the system where the analysis takes place. Therefore, TAU is generally configured and compiled by the user to create custom libraries for use in performance analysis. In addition to generating system specific libraries, this configuration process allows specification of many performance analysis options allowing an extremely diverse range of performance experiments to be carried out with TAU. Each separate configuration operation produces a stub makefile and a library file that is used to compile an instrumented program for analysis.

Instrumentation. TAU's fundamental functionality is based on source code instrumentation. At the most basic level this consists of registering the entry and exit of methods within the program via calls to the performance analysis system. Performance analysis of a given program can be focused on a given set of functions or phases of the program's execution by adjusting which functions are instrumented. A common application of such selective instrumentation is to exclude small, frequently called routines to help reduce performance monitoring

overhead. TAU includes utilities to perform automatic instrumentation of source code. TAU provides compiler scripts which act as wrappers of the compilers described at TAU's configuration. Use of these scripts in place of a conventional compiler results in fully instrumented binary files without modification to the original source.

Analysis. Depending on the configuration settings provided to TAU, it can generate a wide variety of performance data. TAU includes utilities to convert both its profile and trace output to a diverse array of other performance data formats, allowing performance analysis and visualization in many third party performance analysis programs. Performance profiles are automatically uploaded to a PerfDMF [5] repository for analysis.

Additionally, TAU includes its own facilities for analysis of performance data. The ParaProf[11] profile analysis tool, for example, provides a full set of graphical tools for evaluation of performance profile data. PerfExplorer [12] is a multi-experiment analysis and data mining tool, designed to provide parametric study analysis and intelligent analysis of results using performance data, metadata, analysis scripts, and inference rules.

TAU Plugin for Eclipse PTP. Currently, three separate TAU plugins have been developed for Eclipse. Each allows performance analysis within the scope of a different Eclipse IDE implementation, one for the JDT, one for the CDT and one for the PTP. The TAU JDT plugin requires only the standard Eclipse SDK distribution and allows TAU analysis of Eclipse Java projects. The TAU CDT and TAU PTP plugins allow performance evaluation of C and C++ programs within the standard, sequential, C/C++ IDE implementation and the PTP's parallel implementation respectively. Both the TAU CDT and TAU PTP plugins support Fortran when the Photran plugin is installed.

The TAU CDT and PTP plugins extend the CDT and PTP launch configuration systems, respectively. They allow the selection of a TAU stub makefile, which will determine which TAU libraries are used at the program's compilation. The plugins also allow specification of selective instrumentation, other TAU-specific compilation options and data collection options.

When an application is launched within Eclipse using the TAU plugins an instrumented executable will be generated and run using the selected options. This executable will then be run by the CDT or PTP launch management system. When execution is complete profile data may be stored automatically in a local PerfDMF database and viewed in ParaProf. Essentially, once the TAU plugins for the desired IDEs have been installed and configured, obtaining performance data from an Eclipse project is simplified to a sequence of mouse clicks. A complete description of the plugins can be found in [3].

3 Design and Implementation

The initial implementation of the performance analysis work-flow system followed a linear three-step process. Compilation, execution and analysis were performed using the parameters specified by the user in the launch configuration

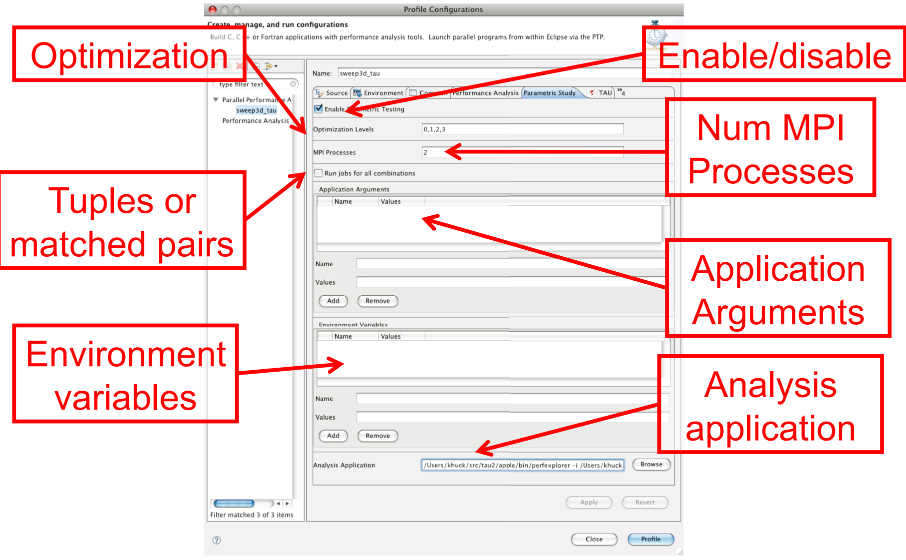
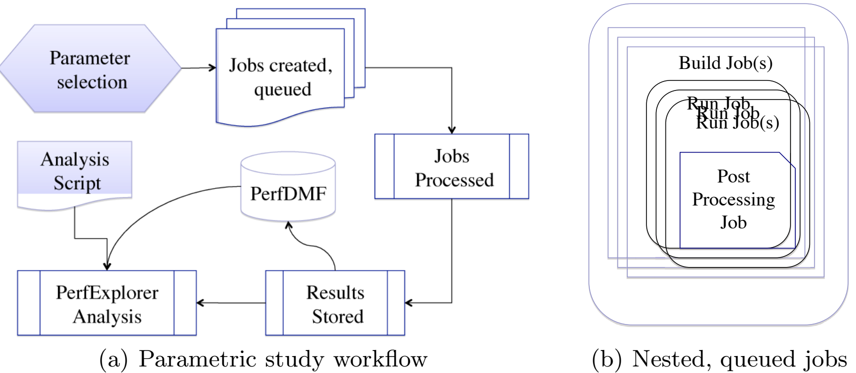


Fig. 1. The Experiment Management user interface

interface. Each step was handled by the job management system provided by the Eclipse API. However the respective jobs made assumptions about their execution order. To add support for multiple parametric combinations it was necessary to make significant changes to the existing work-flow system.

The first such change was to allow the user to specify lists of parameters in a new UI component specific to parametric analysis, as shown in Figure 1. The values available for parametrization are build optimization options, processor count, application arguments and run-time environment variables. Initially we generated one set of parameters for each combination of list elements. Subsequently we added the ability to limit parameter sets to those required such as for weak scaling studies. This was accomplished by creating tuples from the parameter lists, where each parameter list had the same number of potential values. Each index of the parameter lists were matched up (i.e. the first parameter value in each populated list for one experiment, the second parameter value in each populated list for the second experiment, etc.) In the case of weak scaling studies, for each of M processor count values, there would be N input problems (where $M = N$) so rather than generating $M * N$ experiments, there are only N experiments.

The procedure for each parameter set generated is similar to the linear system used in the earlier implementation. However, some modifications were needed to improve efficiency and provide each step with data required from previous steps. For example, because the build and launch steps are independent in the Eclipse environment, only one build operation needs to be performed for each set of build parameters. Each combination of launch parameters is run once on a



(a) Parametric study workflow (b) Nested, queued jobs

Fig. 2. 2(a) shows the overall parametric study workflow. 2(b) shows the nesting of jobs. There is an outer loop of build jobs and an inner loop of run jobs. Each run is followed by a post-processing step, where performance data is uploaded to PerfDMF. After all build and run jobs are complete, the analysis job is run.

single executable generated by a build step. Then the executable is rebuilt with the next set of build parameters and the process repeats.

The single data structure previously responsible for management of the build, execution, and analysis steps was divided into three task-specific components to better support the decoupling of the steps in parametric analysis. Because the launch and analysis jobs relied on data created by the build job, it had been necessary to perform a full build before performing any of the subsequent steps. Thus, another change necessitated by the more flexible parametric analysis system was to initialize the relevant data in the build job object upon its instantiation rather than upon being run. This enables each job to be created and put into a queue before any job is run. The jobs can then be run in succession. A similar dependency existed between the launch job and the post-launch job - the performance data cannot be archived until the execution is complete. These architectural changes will help with the future addition of more complex work-flow capabilities.

The full operating procedure of the parametric analysis system, shown in Figure 2, is as follows. Given the lists parameters from the user, a list of distinct build and launch parameter sets is generated. For each build parameter set in the list, a build job is created and placed in a job queue. After every such build job, for each launch parameter set, launch and analysis jobs are created and placed in the queue. When the queue is fully populated the first job is run. The last action of each job is to initiate the subsequent job in the queue. The final analysis job in the queue contains additional instructions to launch any final analysis operations on the whole of the data generated by the parametric run.

As of this writing the parametric analysis system's support for user specification of build parameters remains rudimentary. The build system provided by Eclipse contains compiler-specific options which must have valid values selected.

Thus, providing a relatively simple UI for specification of arbitrary build parameters requires some foreknowledge of the implementation of Eclipse build-chains for specific compilers. This is in contrast to the launch configuration system where the user may specify arbitrary strings as environment variables or program arguments.

The parametric analysis system was developed to inter-operate directly with the TAU performance analysis system. However the Eclipse performance framework is designed for more general applications. Presently the TAU plug-in is the only component of the performance framework which fully exploits the capabilities of the parametric analysis system. Fully incorporating support for arbitrary analysis tools remains a priority.

4 Examples

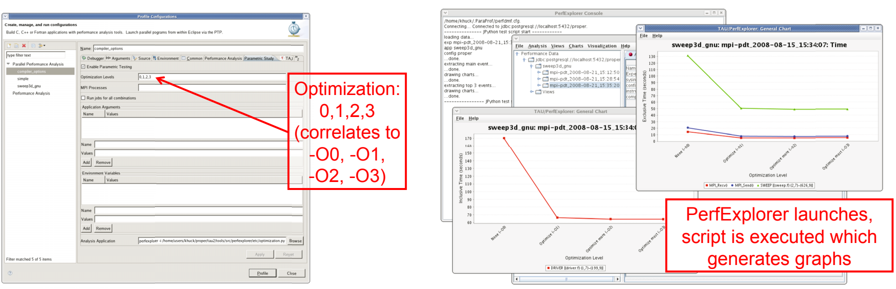
To demonstrate the functionality of the experiment management support we added to Eclipse PTP, we constructed a number of parametric study examples. In this section, we describe two applications, Sweep3D and LU from NPB3.2.1, used in three different parametric studies.

4.1 Sweep3d

Sweep3D [13] solves a 1-group, time-independent, discrete ordinates, 3D Cartesian geometry neutron transport problem. The main algorithm is a wavefront process across the I and J dimensions, and is pipelined along the K dimension. The algorithm gets its parallelism from the I, J domain decomposition. Sweep3D is written in Fortran 77, and uses MPI. There is also a timer routine written in C, but in this experiment, the timer routine was disabled, as we were using TAU for instrumentation. The code was also modified to take the name of the input file as a command line argument, to allow for parametric studies. This was necessary, as the input file also specifies the domain decomposition in each direction, and the total number of processes has to match the number of MPI processes.

The first parametric study was a compiler optimization study. Sweep3D was compiled with the GNU Fortran compiler [14], using four different optimization settings: -O0 (no optimization), -O1 (some optimizations), -O2 (more optimizations), and -O3 (most optimized). The results of the parametric study are shown in Figure 3.

In order to process the experiment, four build jobs were automatically constructed, each with a different compiler optimization setting. Each of the build jobs had a corresponding run job, and a post-run job. The post-run job for each execution was to save the TAU performance profiles into the PerfDMF database for that application, creating a new experiment in the database. After all of the build and run jobs were complete, there was one final post-processing job which ran PerfExplorer and executed an analysis script. The script loaded the performance data into PerfExplorer and generated two charts, one showing the total



(a) Compiler Options selected in Eclipse (b) PerfExplorer output

Fig. 3. The results of running the compiler optimization study - PerfExplorer is executed, and the results of the study are visualized in two charts

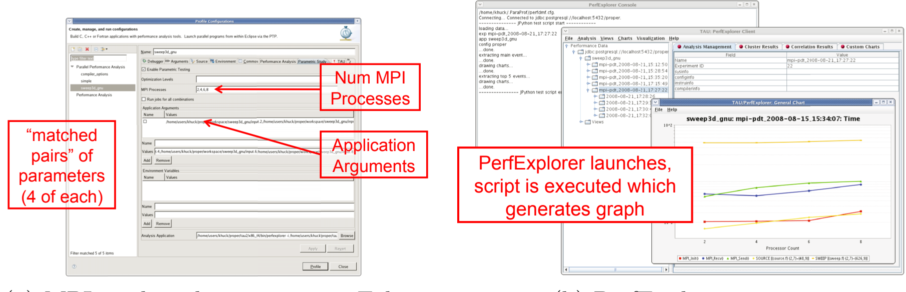
runtime for the application for the four optimization settings, and one chart showing the runtime for the three most time-consuming instrumented regions in the application.

The second parametric study was a weak scaling study. This was an interesting challenge, as for each "number of processors" value, there was a corresponding input problem which scaled at the same rate as the number of MPI processes. In our study, we used 100 grid cells per processor in each direction, and 100 planes in the K direction. Therefore, for 2, 4, 6, and 8 processors, the problem size was 20,000, 40,000, 60,000 and 80,000 total grid cells, respectively. The results of the study are shown in Figure 4.

In order to process the experiment, one build job was constructed, as only one was necessary. Four run jobs were created, each with a different number of MPI processes and a corresponding input file. Again, each of the run jobs had a corresponding post-run job to save the performance profiles to PerfDMF. After the one build and all four run jobs were complete, there was one final post-processing job which ran PerfExplorer and executed an analysis script. The script loaded the performance data and generated one chart showing the runtime for the five most significant instrumented regions in the application. The chart demonstrated that there was a slight increase in overhead in the main computation routines (not uncommon, but ideally the time spent in computation would stay level for all numbers of processors), and an increase in overhead in the MPI communication routines, which is expected in scaling studies.

4.2 NPB 3.2.1 LU

The NAS (NASA Advanced Supercomputing) Parallel Benchmarks (NPB)[15] are a set of programs designed to evaluate the performance of parallel systems. NPB 3.2.1 includes serial, MPI and OpenMP implementations of computational fluid dynamics algorithms. We elected to use the LU benchmark in order to evaluate the OpenMP parallelism functionality in the experiment management system. The LU benchmark uses successive over-relaxation to solve a diagonal system by splitting it into block lower and upper triangular systems.



(a) MPI weak scaling options in Eclipse (b) PerfExplorer output

Fig. 4. The results of running the MPI weak scaling study - PerfExplorer is executed, and the results of the study are visualized in one chart

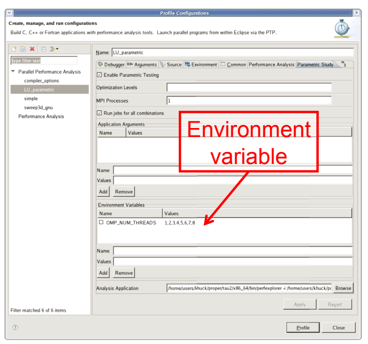
Our third parametric study was a strong scaling study. In this study, we varied the number of threads available to the OpenMP runtime, and compared the results from each execution of the benchmark. In our study, we constructed the "A" class problem, which solves a $64 * 64 * 64$ system in 250 iterations. We requested a study with all integer values of threads between one and eight, inclusive. The results of the study are shown in Figure 4.

In order to process the experiment, only one build job was constructed. Eight run jobs were created, each with a different number of OpenMP threads. This was accomplished by setting the `OMP_NUM_THREADS` variable to a different value for each run. As with the other examples, each of the run jobs had a corresponding post-run job to save the performance profiles to PerfDMF. After the one build and all eight run jobs were complete, there was one final post-processing job which ran PerfExplorer and executed an analysis script. The script loaded the performance data and generated two charts, one showing the total runtime for the application for each of the eight thread values, and one chart showing the runtime for the ten most time-consuming instrumented regions in the application.

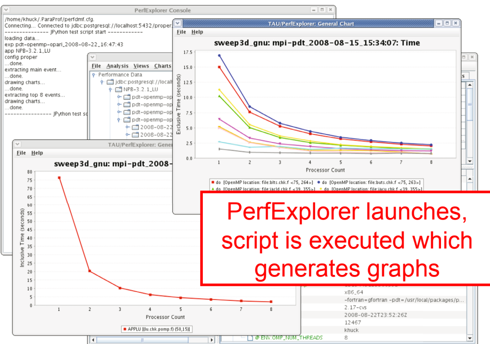
5 Related Work

There are a few related experiment management systems, but to the best of our knowledge, none of them are integrated into an application development environment such as the Eclipse IDE, nor do they support resource managers on large shared systems.

Prophesy [16] is an online database which uploads and stores data from instrumented parallel application runs. Prophesy applies the performance database to manage multi-dimensional performance information for parallel analysis and modeling. The data is accessible from a web interface, and various models can be built, including curve fitting, parametric models, and kernel coupling. There are data generation and submission (of the performance data to the Prophesy database) utilities as part of Prophesy, but they are not automated.



(a) OpenMP strong scaling options in Eclipse



(b) PerfExplorer output

Fig. 5. The results of running the OpenMP strong scaling study - PerfExplorer is executed, and the results of the study are visualized in two charts

Pythia-II [17] is a system for generating performance data from a large parametric space, with the goal of recommending optimized solutions to developers from a number of alternatives. The Pythia-II system combines knowledge discovery with recommender systems to mine performance data, and provide runtime selection of application parameters. While Pythia-II has software for generating performance data, it is for the purpose of searching within the recommender system, and is not intended for general purpose parametric studies.

ZENTURIO [18] is an environment for generating parametric studies for parallel performance analysis. On completion of an application execution, the performance data is automatically stored in a repository. The environment includes a performance visualizer which can perform multi-experiment analysis. ZENTURIO includes support for cluster and Grid computing.

Aksum [19] is a related multi-experiment performance analysis tool which automatically instruments an application, builds and executes the application with given parametric values, and analyzes the results with the goal of locating performance bottlenecks.

6 Conclusion

The work reported here demonstrates initial steps toward integrating automated performance analysis in the Eclipse IDE. In particular, we showed how parametric studies are made easier with Eclipse PTP and Experiment Management support. Support for optimization settings, MPI processor counts, environment variables, and application arguments were developed in Eclipse and used to generate experiments for execution with the TAU Performance System. The project also expanded and improved the automation of analysis results using PerfExplorer scripts.

While successful, there are a few open issues and related problems for our chosen solution. The Eclipse PTP support for launch managers is limited, and not yet robust unless specific versions of supported MPI libraries are used. There are four resource managers supported in Eclipse PTP (ORTE [20], IBMLL [21], PE [22], MPICH2 [23]), but they were either not in use on our parallel systems, or did not work as advertised. For this reason, despite the fact that we had a 128-core system at our disposal, we were unable to perform parametric studies which ran on more than one node. The parametric studies can be performed at large scale on any system that properly supports the PTP. Hopefully, more robust support in the PTP will be provided for more systems in the future.

We are also investigating better ways to handle unusual combinations of parameters. Currently, either all combinations of all parameter values or matched pairs are supported. Other possible ways to specify parameters include value ranges or algorithmic expressions. There are also questions about whether values in the experiment management support override the values in the application build and run configurations, or add to them. Flexible parameter mechanisms would allows the tools to be more broadly applied. In addition, the larger problem of specifying complex combinations of parameters to the build process is as yet unresolved, as described in Section 3.

In conclusion, we believe that as the Eclipse PTP support continues to improve, we will see more parallel application development in Eclipse, and parametric studies submitted from IDEs will become more commonplace.

Acknowledgments

University of Oregon research is sponsored by contracts DE-FG02-07ER25826 and DE-FG02-05ER25680 from the MICS program of the U.S. DOE, Office of Science and NSF grant #CCF0444475.

References

1. The Eclipse Foundation: Eclipse.org Home (2008), `http://www.eclipse.org`
2. Shende, S., Malony, A.D.: The TAU parallel performance system. The International Journal of High Performance Computing Applications 20(2), 287–331 (Summer 2006)
3. Spear, W., Malony, A.D., Morris, A., Shende, S.: Integrating TAU with Eclipse: A Performance Analysis System in a Integrated Development Environment. In: Gerndt, M., Kranzlmüller, D. (eds.) HPCC 2006. LNCS, vol. 4208, pp. 230–239. Springer, Heidelberg (2006)
4. Spear, W., Malony, A.D., Morris, A., Shende, S.: Performance Tool Workflows. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 276–285. Springer, Heidelberg (2008)
5. Huck, K., Malony, A., Bell, R., Morris, A.: Design and Implementation of a Parallel Performance Data Management Framework. In: Proceedings of the International Conference on Parallel Computing (ICPP 2005), pp. 473–482 (2005)

6. Huck, K.A., Malony, A.D., Shende, S., Morris, A.: Scalable, Automated Performance Analysis with TAU and PerfExplorer. In: Parallel Computing (ParCo), Aachen, Germany (2007)
7. The Eclipse Foundation: Eclipse Java Development Tools (JDT) Subproject (2008), http://www.eclipse.org/jdt
8. The Eclipse Foundation: Eclipse C/C++ Development Tooling - CDT (2008), http://www.eclipse.org/cdt
9. The Eclipse Foundation: Photran - An Integrated Development Environment for Fortran (2008), http://www.eclipse.org/photran
10. The Eclipse Foundation: PTP - Eclipse Parallel Tools Platform (2008), http://www.eclipse.org/ptp
11. Bell, R., Malony, A., Shende, S.: A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 17–26. Springer, Heidelberg (2003)
12. Huck, K.A., Malony, A.D., Shende, S., Morris, A.: Knowledge Support and Automation for Performance Analysis with PerfExplorer 2.0. Scientific Programming, special issue on Large-Scale Programming Tools and Environments 16(2-3), 123–134 (2008)
13. LLNL: The ASCI Sweep3D Benchmark (2006), http://www.llnl.gov/asci/purple/benchmarks/limited/sweep3d/
14. Free Software Foundation, Inc.: GNU Fortran - Free Number Crunching FOR All! (2008), http://www.gnu.org/software/gcc/fortran/
15. Bailey, D., Harris, T., Saphir, W., van der Wijngaart, R., Woo, A., Yarrow, M.: The NAS Parallel Benchmarks 2.0. Technical Report Technical Report NAS-95-020, NASA Ames Research Center (December 1995)
16. Taylor, V., Wu, X., Stevens, R.: Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications. SIGMETRICS Perform. Eval. Rev. 30(4), 13–18 (2003)
17. Houstis, E.N., Catlin, A.C., Rice, J.R., Verykios, V.S., Ramakrishnan, N., Houstis, C.E.: PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. ACM Trans. Math. Softw. 26(2), 227–253 (2000)
18. Prodan, R., Fahringer, T.: On Using ZENTURIO for Performance and Parameter Studies on Cluster and Grid Architectures. In: 11th EuroMicro conference on Parallel Distributed and Network-Based Processing, PDP 2003 (February 2003)
19. Fahringer, T., Clovis Seragiotto, J.: Aksum: a performance analysis tool for parallel and distributed applications, pp. 189–208 (2004)
20. The Open MPI Project: Open MPI: Open Source High Performance Computing (2008), http://www.open-mpi.org/
21. IBM: IBM Redbooks — Workload Management with Load Leveler (2008), http://www.redbooks.ibm.com/abstracts/SG246038.html
22. IBM: IBM POWER processor-based servers: Software, Parallel Environment (PE) (2008), http://www-03.ibm.com/systems/p/software/pe/index.html
23. MPICH2: High-performance and Widely Portable MPI (2008), http://www.mcs.anl.gov/research/projects/mpich2/

Trace-Based Analysis and Optimization for the Semtex CFD Application – Hidden Remote Memory Accesses and I/O Performance

Holger Mickler, Andreas Knüpfer, Michael Kluge,
Matthias S. Müller, and Wolfgang E. Nagel

Technische Universität Dresden, Center for Information Services and High Performance Computing (ZIH), 01062 Dresden, Germany
{holger.mickler,andreas.knuepfer,michael.kluge}@tu-dresden.de
{matthias.mueller,wolfgang.nagel}@tu-dresden.de

Abstract. In this paper we present the analysis and optimization of the Semtex CFD application on the basis of trace data obtained with VampirTrace and visualized by Vampir. In the course of the paper the evaluation of I/O performance with regard to globally shared I/O resources and the detection of hidden remote memory accesses with the help of special hardware performance counters will be highlighted.

Keywords: Tracing, Performance Analysis, Remote Memory Access, I/O.

1 Introduction

This paper presents two aspects of the analysis and optimization process of the parallel CFD (computational fluid dynamics) application Semtex on the SGI Altix 4700 platform.

Computation and communication are the most prominent targets for performance improvement. Yet, in this paper we investigate two effects related to remote memory access in a NUMA environment and to extensive I/O activity.

The following Sect. 2 gives an overview of instrumentation and trace collection with VampirTrace and trace visualization with the Vampir and VampirServer tools. Section 3 depicts the checkpointing and communication mechanisms of the Semtex CFD code and presents detailed detection of two interesting performance flaws as well as the successful optimization of both. The paper ends with a short conclusion and outlook.

2 Trace Collection and Visualization with Vampir

VampirTrace is a scalable and portable event tracing software for sequential and parallel applications. It features tracing of applications on UNIX platforms in C, C++ and Fortran supporting MPI, OpenMP and hybrid parallelism. It

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 295–304, 2009.
© Springer-Verlag Berlin Heidelberg 2009

includes support for automatic code instrumentation and a sophisticated run-time measurement library. VampirTrace is developed at the Center for Information Services and High Performance Computing (ZIH) at Technische Universität Dresden in collaboration with the KOJAK project of research center Jülich [1] and is available under a BSD open source license.

Vampir is an interactive trace visualization and analysis tool developed at ZIH, TU Dresden. It allows detailed post-mortem investigation of dynamic parallel run-time behavior as well as statistical summaries of arbitrary intervals of run-time [2,3]. The successor version VampirServer uses a client-server approach with distributed processing of trace data that allows an interactive work-flow for very large data sets [4].

2.1 Performance Counter Support

Even though VampirTrace focuses on event tracing and collecting event-specific information, it utilizes additional statistical information about dynamic run-time performance. Most notably, it supports the PAPI performance counter library, that defines a common interface for reading hardware performance counters [5]. On one hand, it makes common performance counters available with standard names on almost all platforms. This includes the counters for floating point operations or cache misses/hits for various cache levels. On the other hand, PAPI allows to query a huge number of platform specific performance counters that are rarely used. Yet, sometimes such counters allow insight into very special performance issues, as shown in this paper.

2.2 Application Specific and System Wide I/O Tracing

Besides the classical targets of performance analysis computation and communication, another important component of HPC applications is input/output (I/O) from/to files on mass storage systems. In particular, with expanding storage sizes and working sets, the time spent for data accesses on the storage system makes up more and more of the total application run time. Yet, the speed of storage systems does not increase accordingly. Like the so called *memory wall* [6] that inhibits faster computation because of inadequately slow memory accesses there is a similar *input/output wall* for accesses to the storage systems.

This makes the analysis and optimization of the I/O behavior increasingly important. This is especially true for data-intensive applications that scale well with the number of processors. Usually, such codes scale with a constant working set per CPU. Thus the data to be transferred to/from the storage system grows linearly with the degree of parallelism exceeding the I/O capacity eventually.

Therefore, the recent development of VampirTrace contains some approaches for gaining insight into dynamic I/O behavior of parallel applications [7,8]. This includes instrumentation and tracing of application-level I/O calls as well as system-wide I/O throughput of the global SAN (Storage Area Network) infrastructure. The former is important for detailed examination of user-space I/O

requests. The latter is necessary to include the effects from concurrent I/O activities of other applications – the SAN infrastructure cannot be used exclusively like CPU or (parts of) the communication infrastructure (to some extent).

3 Tracing and Analysis of the Semtex Application

Semtex is a parallel CFD code which scales very well over 512 CPUs [9]. It is very data intensive and is used regularly for highly parallel and long-running simulations on the HPC infrastructure of ZIH.

Semtex employs an integrated checkpoint/restart mechanism in order to divide a single simulation into convenient sections that fit well into the batch system policy and make it robust against system failures. Additionally, multiple checkpoints retrieved at small intervals can be used to visualize the simulation. This allows for verification of the correctness and refinement of the simulation, respectively.

The checkpoint mechanism saves the complete parallel working set of a simulation after a given number of time step iterations. For a visualization of the simulation checkpoints are written every 200 iterations. If no visualization is needed, the simulation runs for 5000 time steps (ca. 4h real-time on 128 CPUs) after which a checkpoint is taken that is used as starting point by the next job.

Most of the simulations running at ZIH use a working set suitable for running on 128 processors that results in checkpoints of 5 gigabytes in size. The less often used next larger working set has checkpoints occupying 20 GB of disk space.

As simulations carried out with Semtex account for a large share of the CPU hours used per year at ZIH, its I/O behavior has been subject to closer performance analysis.

3.1 Instrumentation and Tracing of Semtex

For the analysis Semtex was at first instrumented using the automatic compiler instrumentation offered by VampirTrace. A tracing run on 128 CPUs took 2h 16min for 2000 time steps including 10 checkpoints, and the trace data accumulates to 56 gigabytes – including function calls, MPI specific information and extensive I/O records for both, per-process I/O calls as well as system-wide I/O throughput records. This data provides fine-grained information about the application. Figure 1 shows a section of the overall run-time including two checkpoint phases which can easily be identified.

Although VampirServer handles such large traces without problems, the overwhelming details make an analysis cumbersome. The level of detail (and hence the size of trace files) was reduced by employing VampirTrace's filter abilities which allow to record only a given number of calls per function or to skip certain functions completely.

Nevertheless, the overhead caused by the automatic instrumentation still was very large – tracing time was about 2.5 times of original runtime.[1] As a

[1] This is a known problem when compilers have to decide whether to instrument or inline a function – the Intel compilers used in this example favor instrumentation over inlining, therefore much performance is lost.

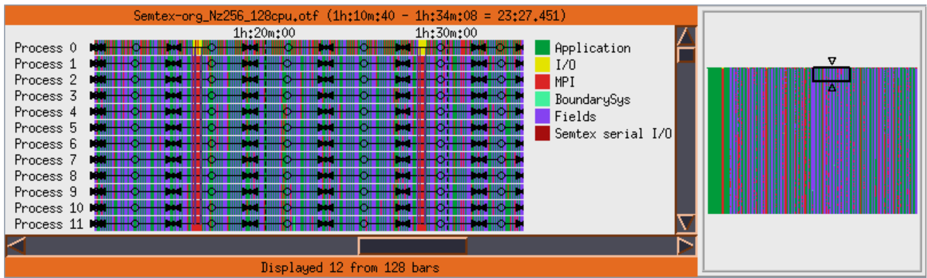


Fig. 1. Vampir's Process Timeline display for a typical execution of Semtex

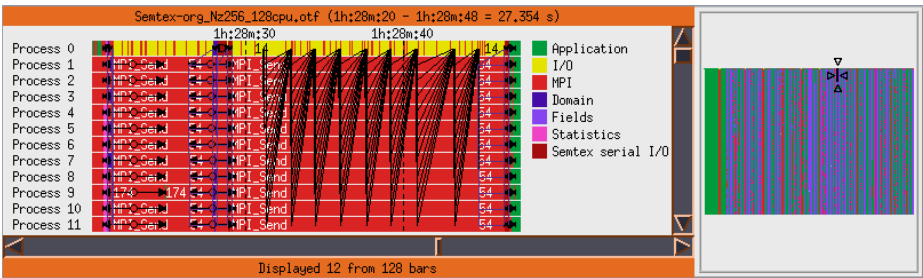


Fig. 2. Timeline display zoomed to a single checkpoint phase

consequence, the code was manually instrumented and afterwards, the tracing introduced only a marginal overhead.

However, the automatic instrumentation was by no means useless. The available tracing data delivered invaluable insights into the workflow of Semtex that were used to quickly identify interesting source code locations where instrumentation calls have been inserted.

3.2 I/O-Related Performance Bottlenecks

For investigation of I/O related performance we looked closer on the checkpoint phases in between the time step simulation. Figure 2 shows a zoomed time line for a typical checkpoint phase. Process 0 is dominated by I/O activity shown in yellow while the remaining processes spend this time in MPI calls. This is a classic *single writer* situation, where one process is collecting all data from its peers via message passing in order to write it to the file system.

This scheme is obviously unfavorable for a massively parallel program and limits the otherwise good performance. In particular, it inhibits scalability as checkpoint phases will grow linearly for growing CPU counts.

Further investigating the I/O behavior of Process 0 revealed additional performance problems. All checkpoint phases turned out to follow a very regular pattern of receiving and writing constant sized blocks of approximately 1.4 MB

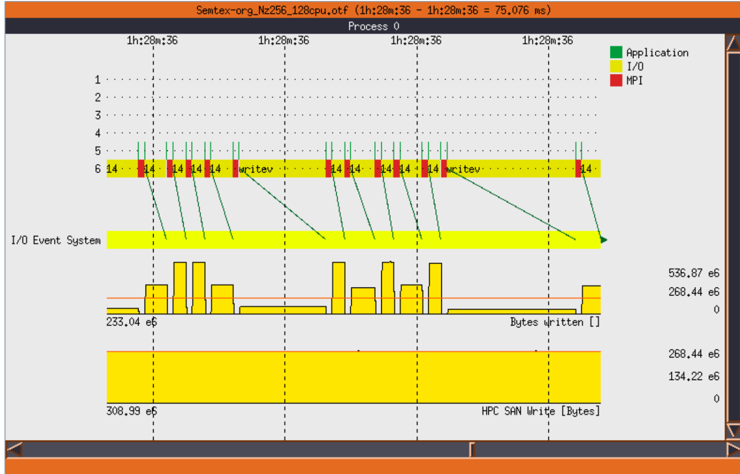


Fig. 3. Regular pattern of MPI receive (red) and write (yellow) activities with notable variation in throughput. The upper counter shows the process-related write speed while the lower one shows the global SAN throughput.

as shown in Fig. 3. Yet, the single write operations show quite differing speed as shown in the *bytes written* performance counter, compare Sect. 2.2.

When inspecting I/O performance of any single process it is important to consider the current utilization of the SAN infrastructure. Usually, the I/O network is not exclusively used by an application but globally shared. Therefore, the effect could have been caused from outside, i.e. any other application with extensive I/O utilization. This is not the case in this example. The comparison of the local counter *bytes written* and the global one *HPC SAN write* in Fig. 3 shows the same I/O throughput on average. Furthermore, the global I/O speed, which is only available with a resolution of one sample per second (compare Sect. 2.2), is almost constant during all checkpoint phases.

3.3 Optimization of I/O Performance

Our analysis of I/O behavior revealed two bottlenecks: On one hand, the *single writer* problem, and on the other hand the fluctuating local write speed.

Both of these problems were solved by modifications to the checkpoint code. From the trace analysis we learnt the following facts:

- The speed of single write calls of constant size is heavily fluctuating. Even though most calls are fast, the regularly occuring very slow calls destroy the over-all I/O performance – compare peaks in *bytes written* counter in Fig. 3.
- The speed of SAN I/O is the same during all checkpoint phases and near zero just before and after these phases which confirms the reliability of the application's I/O measurements.

– The time step iteration is interrupted by checkpoint phases from time to time. Therefore, every I/O phase is followed by a computation phase without I/O activity – see Fig. 1.

This led to the following hypothesis: The delays in I/O API calls on the application level are caused by the caching I/O subsystem of the operating system. Two different solutions are available to eliminate this problem. The first is using direct I/O which bypasses the operating system's file cache. Yet, it is rather difficult to implement as special alignments and request sizes have to be used. The second solution does not suffer from those restrictions. Since the write-only scheme of checkpointing does not require instant write to disk, asynchronous I/O can be used. This has the advantage of decoupling the I/O calls, that happen during the checkpoint phases, and the actual I/O operations issued by the operating system, that may be performed concurrent to the following computation phase. Furthermore, with modifying the checkpoint routine so that each process writes its data on its own to the checkpoint file, the single writer problem is addressed as well.

Table 1. Comparison of original and optimized Semtex checkpoint phases

# CPUs	Checkpointing Time (% of total runtime)				Improvement
	Original version		Optimized version		
8	12.1 s	(1.3%)	6.3 s	(0.9%)	47.9%
128	106.8 s	(8.4%)	35.5 s	(3.9%)	66.8%
256	381.7 s	(12.8%)	107.7 s	(5.1%)	71.8%

Table 2. Comparison of checkpoint times, *intermediate checkpoint phases only*

# CPUs	Checkpointing Time (% of total runtime)				Improvement
	Original version		Optimized version		
8	7.5 s	(0.8%)	3.1 s	(0.4%)	58.7%
128	64.7 s	(5.1%)	15.8 s	(1.8%)	75.6%
256	249.3 s	(8.4%)	37.0 s	(1.7%)	85.2%

Based on this hypothesis the checkpoint code of the Semtex application was modified to use asynchronous MPI I/O functions. This yielded an improvement in checkpointing speed of up to 85%, compare Tables 1 and 2.

Table 1 shows the times spent for checkpointing including the final checkpoint. There the gain from asynchronous I/O is not as large as within intermediate checkpoints (see Table 2). This comes from the fact that the application must wait for completion of the I/O within the last checkpoint whereas this is not necessary for intermediate checkpoints.

Table 2 further shows that the proportionate time needed for intermediate checkpoints does not grow when increasing the number of processes from 128 to 256 (1.8% to 1.7%) which underlines the potential of asynchronous I/O.

3.4 Performance Problems in Memory Copy Operations

In the course of the performance evaluation of Semtex's I/O activities, an exceptional high fraction of almost 40% was observed to be spent for communication during time step iterations. Figure 4 shows the profile of one such iteration. One time step needs 1.5 seconds to complete from which 0.59 seconds are used for data exchange. A look into the source code revealed that besides MPI functions, the only other time-consuming calls could be those to `memcpy` which were then enclosed by tracing calls for making them available for analysis (already included in Fig. 4).

Fig. 4. A major part of one simulation step is spent on data exchange, which `memcpy` is responsible for, besides MPI

Further investigation showed that the speed of `memcpy` calls is not constant. On Process 0 the first call is faster than the remaining ones, compare Fig. 5. The same happens for Process 1 whereas on Processes 2 and 3, the third call is faster.

Apart from the difference in speed the copy operations themselves are surprisingly slow. Each `memcpy` call copies 22 kBytes of data in 35 μs (fast case) or in 165 μs (slow case) resulting in transfer rates of 640 MB/s and 130 MB/s, respectively – the underlying architecture would allow for much more.

We found a combination of three phenomena responsible for these effects:

1. Glibc's `memcpy` is not tuned for the Itanium architecture which causes the copy operations to be rather slow.
2. *Single-copy transfers* of SGI's Message Passing Toolkit (MPT, the MPI library in use) introduce hidden remote memory accesses.
3. The different speeds of memory copy operations arise from characteristics of the Altix 4700 architecture in conjunction with the second phenomenon.

To understand the latter, both the workflow of the data exchange code and the architecture of the Altix 4700 have to be taken into account. The Semtex application uses a special data exchange scheme between all processes, which is performed multiple times during every time step iteration of the simulation. It is implemented as follows:

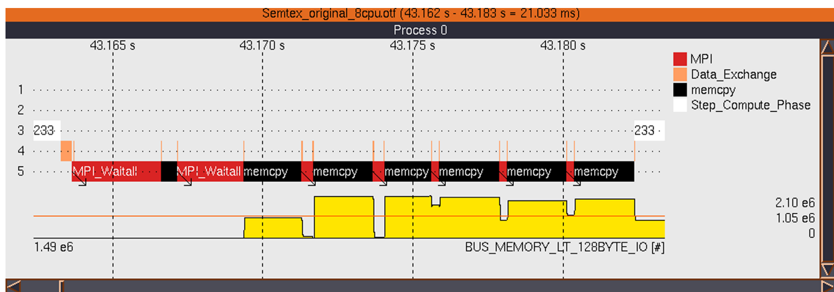


Fig. 5. The Itanium's hardware performance counter indicates unoptimized memory accesses from remote processors

1. Allocate a temporary buffer for data exchange.
2. For each of the other processes do the following:
   – Send own buffer to partner via `MPI_Isend`.
   – Receive data from partner via `MPI_Irecv` into the temporary buffer.
   – Wait for completion of the communication via `MPI_Waitall`.
   – Copy contents of temporary buffer to own buffer.

Therefore, Process 0 exchanges data with Process 1 first, then with Process 2, 3, etc. The sequence is analogous for Process 1. Process 2 communicates first with Process 0, then with Process 1, 3, 4, etc.

Figure 5 shows the details of one complete exchange on Process 0 for a small run with 8 CPUs. The send-receive-copy pattern is executed for every peer process, i.e. seven times in this example.

From this algorithm and the architecture of the Altix 4700, which consists of dual-core Intel Itanium 2 processors[2], we can conclude that the faster `memcpy` calls belong to the data exchange happening between the cores of one CPU.

The findings so far suggest that the `memcpy` calls after communication with processes located on other CPUs access remote memory and are therefore slower. This is somewhat counter-intuitive because the code looks as if the copy routine would access two local buffers – the temporary one and the permanent one.

Yet, the behavior can be explained considering the second phenomenon. For saving memory bandwidth, MPT maps the communication buffers into the peer's address space and then uses a single copy operation to transfer the contents to the destination buffer [10]. Unfortunately, the memory pages of the receiver's temporary buffer seem to be located at the remote party after this operation.

Evidence for this hypothesis is given by a special hardware performance counter of the Itanium 2 processor. Below the time line, Fig. 5 shows the rate of `BUS_MEMORY_LT_128BYTE_IO`, which counts the number of less than full cache line[3] transactions from remote parties. The counter shows a high rate indicating excessive remote memory accesses that transfer less than 128 bytes. Those

[2] At ZIH, in particular, there is one CPU per system board.
[3] L2 and L3 cache line size is 128 bytes on the Itanium 2 processor.

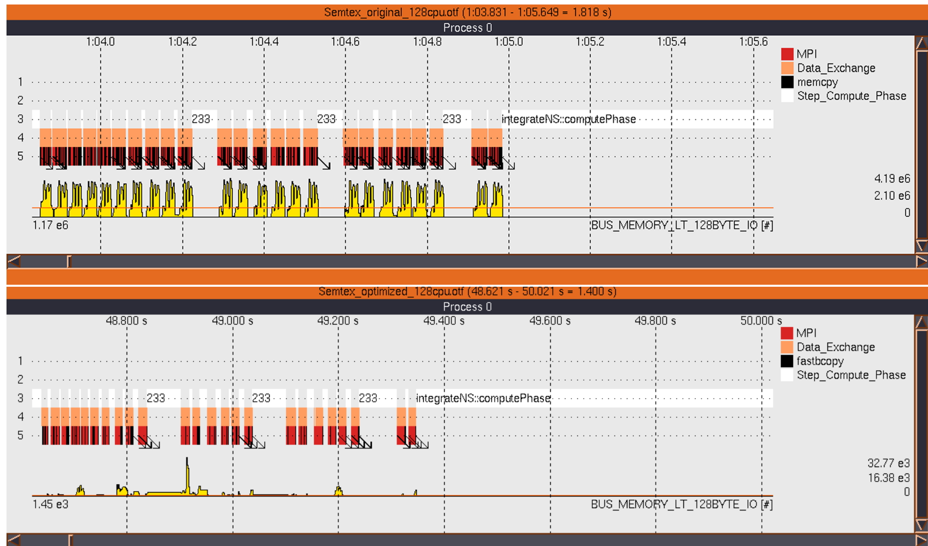


Fig. 6. Comparison of the original data exchange pattern (top) with the optimized counterpart (bottom) for Process 0 for the 20'th time step iteration. Note the much smaller duration of the optimized version (1.4s vs. 1.8s) and the significantly reduced value of the counter showing small-sized remote memory accesses.

have double negative effects on transfer speed. Firstly, smaller transfer sizes mean more remote accesses are needed for copying the same amount of data, and secondly, each remote access suffers from higher latencies compared to local memory accesses. This in turn leads to the poor performance of `memcpy` when it eventually accesses remote memory.

For Process 0, the counter is low for the first `memcpy` belonging to the data exchange with Process 1, and rises afterwards indicating ineffective accesses to remote memory caused by `memcpy`. Looking at Process 2 (not shown here), the third `memcpy` shows no peak, and this scheme continues to the last process.

3.5 Optimization of Communication Scheme

The performance of the data exchange code could be dramatically improved by replacing the `memcpy` calls with the highly optimized `fastbcopy` call available from MPT. This routine achieves copying speeds of 4500 MB/s for local memory and 1300 MB/s when remote memory is involved.

A comparison of the original version with the optimized one is shown in Fig. 6. The architectural optimization of the `fastbcopy` routine is depicted by the `BUS_MEMORY_LT_128BYTE_IO` counter which shows very low rates in the optimized version (lower picture). Usage of this routine doubled the speed of data exchange which allowed for more than 20% improvement in run-time per time step iteration. Together with the I/O optimization, the total runtime of Semtex was reduced by 25%.

4 Conclusion and Outlook

This paper presented two interesting aspects of the performance analysis process for the Semtex application. The proposed optimization steps were confirmed with notable performance improvements for this application. Since this code is used for long-term simulations with large degree of parallelism on the HPC resources of ZIH, TU Dresden, the optimization accounts for a substantial number of CPU hours saved!

Further work will focus on the I/O tracing components of VampirTrace, in particular the availability of system-wide I/O monitoring in a platform independent manner.

References

1. Wolf, F., Mohr, B.: KOJAK - A Tool Set for Automatic Performance Analysis of Parallel Applications (Demonstrations of Parallel and Distributed Computing). In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 1301–1304. Springer, Heidelberg (2003)
2. Nagel, W.E., Arnold, A., Weber, M., Hoppe, H.C., Solchenbach, K.: VAMPIR: Visualization and Analysis of MPI Resources. In: Supercomputer 63, vol. XII(1), pp. 69–80 (1996)
3. Brunst, H., Winkler, M., Nagel, W.E., Hoppe, H.-C.: Performance optimization for large scale computing: The scalable VAMPIR approach. In: Alexandrov, V.N., Dongarra, J., Juliano, B.A., Renner, R.S., Tan, C.J.K. (eds.) ICCS 2001. LNCS, vol. 2074, pp. 751–760. Springer, Heidelberg (2001)
4. Brunst, H., Nagel, W.E.: Scalable Performance Analysis of Parallel Systems: Concepts and Experiences. In: Joubert, G.R., Nagel, W.E., Peters, F.J., Walter, W.V. (eds.) Parallel Computing: Software, Alghorithms, Architectures Applications, pp. 737–744. Elsevier, Amsterdam (2003)
5. PAPI group: Papi - performance application programming interface, http://icl.cs.utk.edu/papi/
6. Wulf, W.A., McKee, S.A.: Hitting the memory wall: Implications of the obvious. Computer Architecture News 23(1), 20–24 (1995)
7. Müller, M.S., Knüpfer, A., Jurenz, M., Lieber, M., Brunst, H., Mix, H., Nagel, W.E.: Developing Scalable Applications with Vampir, VampirServer and VampirTrace. In: Bischof, C., Bücker, M., Gibbon, P., Joubert, G., Lippert, T., Mohr, B., Peters, F. (eds.) Parallel Computing: Architectures, Algorithms and Applications. Advances in Parallel Computing, vol. 15, pp. 637–644. IOS Press, Amsterdam (2007) ISBN 978-1-58603-796-3
8. Mickler, H.: Kombinierte Messung und Analyse von Programmspuren und systemweiten I/O-Ereignissen. Master's thesis, Technische Universität Dresden (2007)
9. Blackburn, H.M., Sherwin, S.J.: Formulation of a Galerkin spectral element-fourier method for three-dimensional incompressible flows in cylindrical geometries. J. Comput. Phys. 197(2), 759–778 (2004)
10. SGI: Linux Application Tuning Guide, http://techpubs.sgi.com/library/manuals/4000/007-4639-008/pdf/007-4639-008.pdf

Scalasca Parallel Performance Analyses of PEPC

Zoltán Szebenyi[1,2], Brian J. N. Wylie[1], and Felix Wolf[1,2]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Germany
[2] Aachen Institute for Advanced Study in Computational Engineering Science, RWTH Aachen University, Germany
{z.szebenyi,b.wylie,f.wolf}@fz-juelich.de

Abstract. PEPC (Pretty Efficient Parallel Coulomb-solver) is a complex HPC application developed at the Jülich Supercomputing Centre, scaling to thousands of processors. This is a case study of challenges faced when applying the Scalasca parallel performance analysis toolset to this intricate example at relatively high processor counts. The Scalasca version used in this study has been extended to distinguish iteration/timestep phases to provide a better view of the underlying mechanisms of the application execution. The added value of the additional analyses and presentations is then assessed to determine requirements for possible future integration within Scalasca.

Keywords: Parallel/distributed systems, performance measurement & analysis tools, application tracing & profiling.

1 Introduction

PEPC [7] is a 3-dimensional particle simulation code which employs a hierarchical, parallel tree algorithm implemented using MPI to compute the forces on the particles. The code is presently used for various applications in plasma physics and astrophysics. According to the author of the code, potential bottlenecks lie in the domain decomposition routine, tree construction and tree 'walk', the last of which requires significant point-to-point communication of multipole information between processors, and is thereby sensitive to load imbalance.

This work presents Scalasca measurements and analyses of the application on the IBM BlueGene/P system of the Jülich Supercomputing Centre. The Scalasca toolset [1,2] is a highly scalable performance analysis toolset capable of both taking runtime summaries or collecting and automatically analyzing event traces. The latter can be searched for complex event patterns which may indicate important performance bottlenecks. Scalasca performs trace analysis in parallel on the same number of processors that was used to originally take the measurement.

In this work PEPC is analyzed using an extended version of Scalasca [6], providing the additional capability of phase instrumentation. This means that after manually identifying the main time-stepping loop of the application and inserting markers around the loop body, subsequent Scalasca measurements become aware of individual iterations and support the analysis of the time-dependent

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 305–314, 2009.
© Springer-Verlag Berlin Heidelberg 2009

behavior or of individual iterations, which corresponds to dynamic phases in TAU [4]. After considering examples of such PEPC execution analyses and presentations, these are assessed to determine requirements for the potential future integration of the corresponding capabilities with Scalasca.

2 Experimental Results

A 1024-way test case, which was run for 1,300 timesteps on a BlueGene/P system, is used here as an example to show what we have learned about the performance characteristics of the application using the tools provided by the extended Scalasca toolset. This example was chosen as it shows not only interesting patterns of time-varying behavior, but also performance problems, like serious load imbalance growing rapidly over time.

Figures 1&2 show an example of the usefulness of having analysis data individually for all the iterations instead of having just a summary of the whole program execution. Figure 1 shows the case where the iterations are not distinguished and only aggregate metrics are available. When looking at the point-to-point communication metric, a few processes appear as hot-spots in the topology pane, showing that there is some imbalance. This is important to recognize, however, additional insight can be extracted from the extended, phase-instrumented analysis. When selecting in turn the individual iterations distinguished in Fig. 2, the execution behavior can be observed evolving over time. During the first iterations, the communication is relatively balanced and there are no extreme hot-spots visible. Over time some hot-spots appear and become increasingly pronounced, and they move at different rates from one MPI rank to the next. The number of hot-spots diminishes towards the end of the 1,300 steps, however, the severities of the highest ones are increasing rapidly.

Figure 3 shows four different views of the same metric, *point-to-point communication count* (i.e., the number of sends and receives), to compare the different kinds of information they provide. The *Phase graph* (upper left) gives an overview of the evolution of the values over time. The meaning of the different colors is as follows: for each timestep, the value for the process with the largest time is shown in red, the median is shown in blue, and the smallest is shown in green. Looking at this graph it is obvious that there is a serious communication imbalance in the application, as the minimum and median values are relatively low and constant throughout the execution, but the maximum value is growing rapidly, and it rises many times higher than the minimum or median values. This means that within each timestep most of the processes are behaving in a relatively balanced fashion, but that there are a few processes being involved in many more communications.

Which processes are responsible for the imbalance? On the *Process graph* (upper right), the x-axis shows the different process ranks, so the different colors now distinguish the minimum, median and maximum value during the different timesteps for the given process rank. It is not obvious which process is responsible as many processes show very high values, but again these can only be for a few

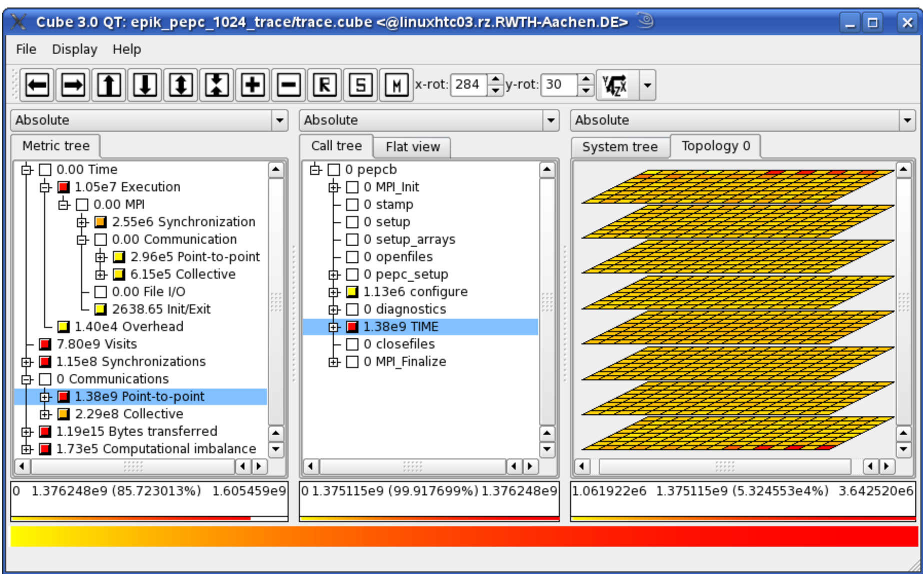


Fig. 1. Scalasca analysis report explorer showing a PEPC runtime summary with significant imbalance in the number of point-to-point communications (selected in left pane). The 1,024 application processes are arranged according to the BlueGene/P physical network topology (right pane).

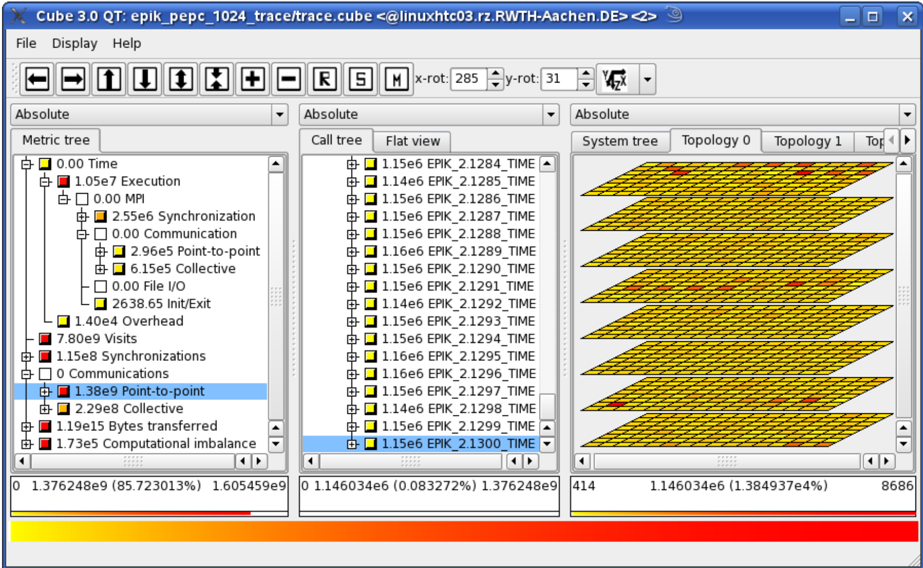


Fig. 2. Scalasca analysis report explorer showing a PEPC runtime summary with timesteps distinguished via phase instrumentation (centre pane)

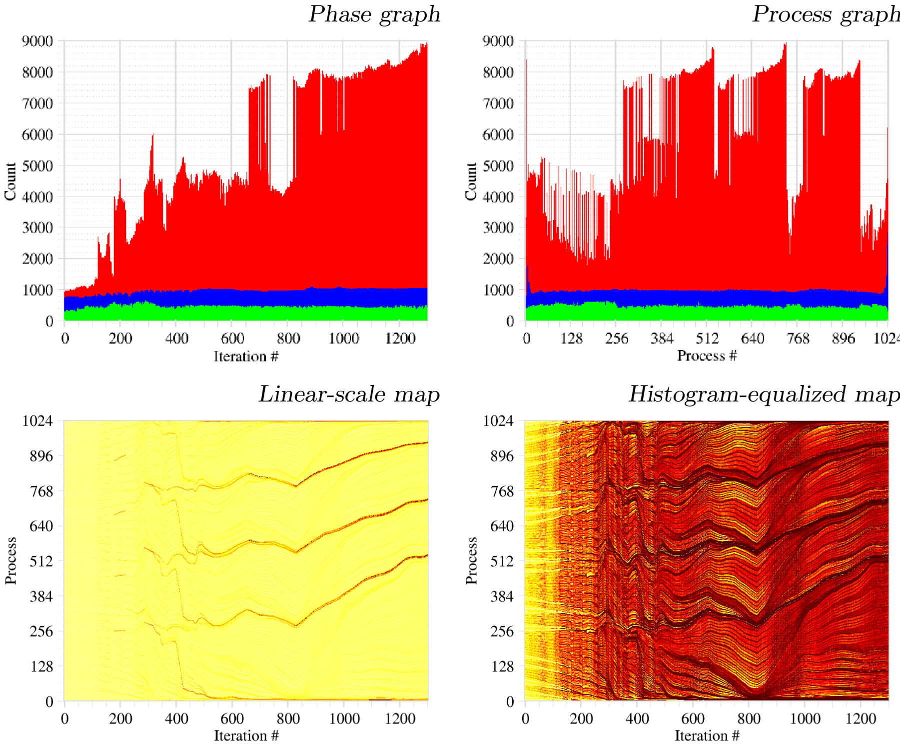


Fig. 3. Different analysis presentations of point-to-point communication count metric

timesteps, as the minimum and median values are low for all of them (except for the first and last few ranks where the median is somewhat higher).

The case is getting increasingly confusing, and we still do not see what is going on here with these high values, but the *Linear-scale map* on the lower left makes it much clearer. On a linear-scale map, the x-axis shows the iteration number, the y-axis the process rank, and the values are color-coded from light yellow (for the lowest value) to dark red (for the highest value, here 9000). The map shows that at the beginning all processes start off relatively balanced, however, after a few hundred timesteps a low number of hot-spots gradually appear whose values get much higher over time than the average. What is interesting about these hot-spots is that they are not bound to any given process, but rather they move to neighboring processes in a systematic and coordinated manner. As they migrate some hot-spots appear to merge, so after around 700 timesteps only five hot-spots remain, each consisting of a few processes.

This movement of the hot-spots is responsible for the confusing values seen in the *Phase* and *Process* graphs, so understanding their behavior in more detail is useful. Taking a closer look at the light yellow area of the *Linear-scale map* reveals that the values in the background are not exactly the same. There are also some patterns there, but they are not very easy to see as their differences

are small compared to the range of the graph. On the *Histogram-equalized map*, light yellow and dark red still mean the same lowest and highest values as on the *Linear-scale map*, but here the histogram of the map values is equalized so that every color level is used for approximately the same number of points. This produces the maximum contrast on the map and reveals previously invisible details: there are many more systematic details down to the finest granularity than there are visible on the *Linear-scale map*.

Figure 4 shows more metric graphs of the test case execution, including the time taken by different activities and the number of bytes transferred in each timestep. Where no explicit legend is given, the different colors have the same meaning as before.

The *Execution Time Breakdown* graph uses a different coloring where yellow is the average time each process spent in pure computation (i.e. non-MPI functions), the small green part is the useful time spent in MPI communication, orange is the blocking time in situations like late sender, and magenta is synchronization time at barriers. These values combine to the total execution time of each timestep, averaged over all the processes. Execution time does not show any differences between processes as execution times are synchronized every timestep due to the collective communications.

The high peak values every 100th iteration are due to checkpointing. Also notable on this graph is the evolution of the total execution time per iteration along the 1,300 timesteps, gradually increasing more than twofold from around 5.5 seconds to more than 12 seconds. From the breakdown this is seen to be due to the computational workload itself growing over time (as the pure computation time is growing in the same way), however, MPI blocking and synchronization times are growing as well.

On the *Execution Time Proportions* graph, the same data is normalized for each iteration to show the fraction of execution time spent in each activity. This graph shows that the proportion of pure computation time is shrinking from around 78% down to 73%, while the proportion of MPI blocking time grows from 3% to 6% and the proportion of MPI synchronization grows from 18% to 19%. MPI blocking time is therefore the fastest growing problem, even though the time spent in synchronizations is still higher.

The *Point-to-point Communication Time* graph shows the load imbalance very clearly, as the median and maximum times spent in point-to-point communications are much higher than the minimum, and the median is around halfway between the minimum and maximum. This means that the imbalance in point-to-point communication time involves many or most of the processes. Comparing with *Point-to-point Late Sender Time*, most of the time is actually spent in situations where the receiver blocked waiting for the corresponding send to be initiated, which suggests that some point-to-point messages are not sent in time. This causes a majority of processes to spend more time in point-to-point communication than is absolutely necessary (which is around the minimum value shown in green).

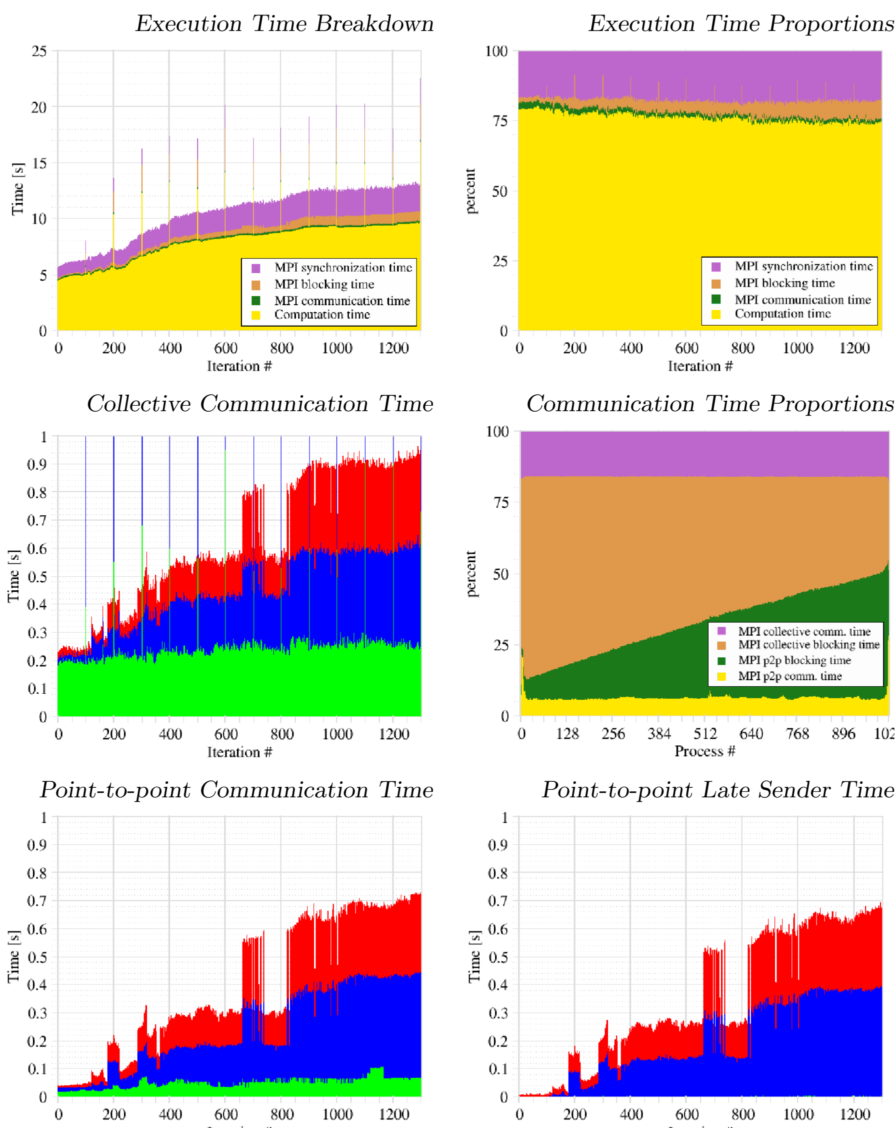


Fig. 4. Graphs of time and bytes transferred in different communications in PEPC

The *Collective Communication Time* graph also shows an imbalance very much like that seen on its point-to-point counterpart, however, with a much higher minimum. Apparently, the minimum time for collective communication in each timestep is longer than the corresponding point-to-point time, but everything in excess of this minimum closely resembles the point-to-point late sender time. The high peaks every 100th iteration are due to checkpointing activity in those timesteps.

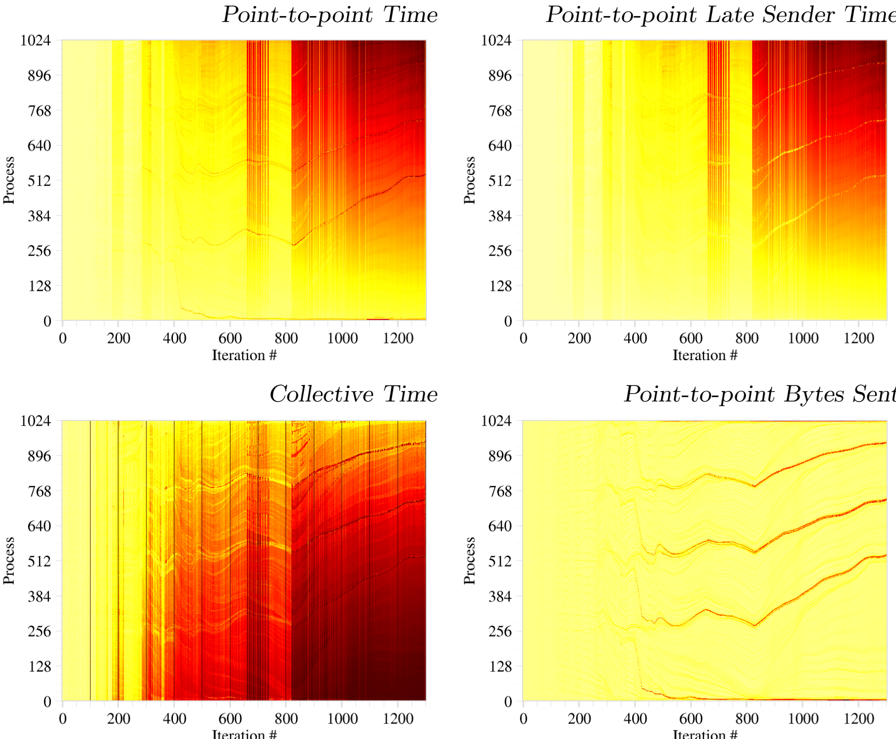


Fig. 5. Maps of time and bytes transferred in different communications in PEPC

Figure 5 shows some maps that help clarify the nature of the communication time imbalance. The *Point-to-point Time* map shows how the above mentioned communication time imbalance is distributed among the processes in a very specific and systematic way. Generally the higher the MPI process rank, the more time it spends in point-to-point communication. There are some exceptions to this rule, as the processes which send more point-to-point messages take somewhat longer than their neighbors, particularly the hot-spots which all have very high communication times.

The *Point-to-point Late Sender Time* map shows a rather similar distribution, with one major difference being the hot-spots. These show very low waiting time in late sender situations, but they have a very high amount of time spent in point-to-point communication. This means that they really communicate a lot and then do not spend much time waiting, while all the others that must wait for them do.

The *Collective Time* map also shows something interesting. It seems to be the inverse of Point-to-point time, in the sense that the order of MPI ranks is reversed here. The higher the MPI rank, the less time it spends in collective communications. This suggests that the price of the communication imbalance in point-to-point communications must be paid again in collective communications. First point-to-point

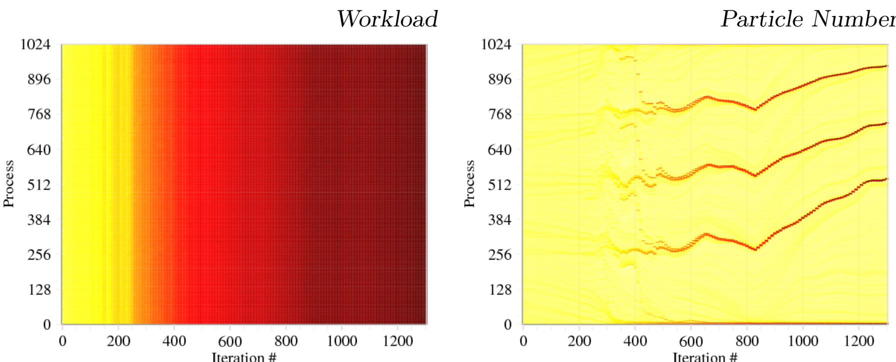


Fig. 6. Maps of application-specific metrics from log files written every 10th timestep

communication causes the processes to get out of balance, then the next collective communication synchronizes them again, and the processes which wait less in point-to-point and are slightly ahead of the others have to wait more while the ones that are later wait less. So in the end they accumulate the same amount of blocking time, which, however, is distributed in reversed fashion across the processes.

This phenomenon is clearly shown in the *Communication Time Proportions* graph in Fig. 4, which shows the process ranks on the x-axis, and the proportion of MPI point-to-point communication time (yellow), point-to-point blocking time (green), collective blocking time (orange) and collective communication time (magenta) on the y-axis. The most important aspect of this graph is the diagonal border between point-to-point blocking time and collective blocking time. The higher the MPI rank, the more point-to-point blocking time and the less collective blocking time was diagnosed, with both waiting time categories consuming around 80% of the total communication time. This highlights how seriously this problem degrades communication efficiency.

But what causes the imbalance? Examining the *Point-to-point Bytes Sent* graph in Fig. 5 it is clear that the hot-spot processes are sending much more data than any other process. *Point-to-point Bytes Sent* and *Received* graphs (not shown due to space restrictions) reveal that while all processes receive around the same amount of data, the amount varies considerably between senders. Therefore the hot-spot processes send much more data than other processes, and they send data to all. This suggests that the hot-spot processes are the communication bottleneck, as they send much more data than the others. Moreover, the data are sent in process rank order, such that higher-ranked processes have to wait for them to complete sending data to lower ranked processes before they receive any data themselves.

Figure 6 shows maps of two application-specific metrics extracted from application log files written every 10th timestep. On the *Workload* map we see the workload metric calculated by the application. According to the developers, PEPC runs a workload-balancing algorithm every timestep where it balances this metric, therefore it is no surprise that it really is balanced over the processes. It further

shows some growth over time which correlates to the growing computational part of the execution time. On the *Particle Number* map we see the number of simulated particles assigned to each process. This means that the workload-balancing algorithm assigns a very large number of particles to the hot-spot processes which in turn causes a communication bottleneck to appear on those processes and leads to the communication imbalance.

The TAU Paraprof 3D visualizer [4] is a third-party tool able to visualize Scalasca analysis reports. 3D visualization is a promising complement to 2D maps, as it can be easier to compare scales of data at different positions by comparing the height of bars as opposed to comparing the brightness of colors.

3 Conclusion

In our analysis of the PEPC execution on BlueGene/P we have identified some complex performance patterns, and have found a good potential for communication performance improvement. The PEPC developer team is actively working on finding the causes that led to the serious imbalance in particle numbers that we identified as the root cause of the point-to-point communication problem. They are also looking at ways to modify the workload-balancing algorithm to avoid this situation.

The depth of analysis we conducted in this case would not have been possible without our prototype Scalasca extensions, namely phase instrumentation and the different visualization techniques used to make the huge amounts of data collected accessible to the user. During the course of the analysis, we have found all the different kinds of visualization techniques (phase and process graphs, linearly-scaled and histogram-equalized 2D maps, 3D visualization) useful in many ways, as the insights they provide often complement each other.

We have also found that there are serious limitations concerning the visualization of the huge amounts of data collected, as it can easily happen that on the monitor or printer used for displaying the data, it is not possible to get sufficient resolution to have one pixel for each process or iteration. Other groups also identified this issue [8,9] and found different solutions, like zooming in on the visualized data while also displaying a miniature map of the whole chart marking the zoomed region, or letting the user choose from different display options when there is more than one data point for a single pixel, such as the maximum, minimum, sum, or median of the values. These techniques can provide partial solutions to the problem, but further investigation of this topic could also prove to be valuable in the future.

Visualizing time-dependent behavior with an animation, where the user can step through iterations to track changes of a metric over time, also seems to be an interesting possibility that proved useful in understanding PEPC measurement results. Furthermore, 3D visualization such as the one offered by TAU was found to be extremely valuable and perhaps the most insightful of all the different visualization techniques, but also the technically most challenging.

References

1. Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Frings, W., Fürlinger, K., Geimer, M., Hermanns, M.-A., Mohr, B., Moore, S., Pfeifer, M., Szebenyi, Z.: Usage of the SCALASCA toolset for scalable performance analysis of large-scale parallel applications, in Tools for High Performance Computing. In: Proc. 2nd Int'l Workshop on Tools for High Performance Computing, Stuttgart, Germany, pp. 167–181. Springer, Heidelberg (2008)
2. Jülich Supercomputing Centre, Scalasca toolset for scalable performance analysis of large-scale parallel applications, `http://www.scalasca.org/`
3. Wylie, B.J.N., Wolf, F., Mohr, B., Geimer, M.: Integrated runtime measurement summarization and selective event tracing for scalable parallel execution performance diagnosis. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, pp. 460–469. Springer, Heidelberg (2007)
4. Malony, A.D., Shende, S.S., Morris, A.: Phase-based parallel performance profiling, In: Parallel Computing: Architectures, Algorithms and Applications. Proc. 11th ParCo Conf., Málaga, Spain, October 2006. NIC Series, vol. 33, pp. 203–210. John von Neumann Institute for Computing, Jülich (2006)
5. Fürlinger, K., Gerndt, M., Dongarra, J.: On using incremental profiling for the performance analysis of shared-memory parallel applications. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 62–71. Springer, Heidelberg (2007)
6. Szebenyi, Z., Wylie, B.J.N., Wolf, F.: SCALASCA parallel performance analyses of SPEC MPI2007 applications. In: Kounev, S., Gorton, I., Sachs, K. (eds.) SIPEW 2008. LNCS, vol. 5119, pp. 99–123. Springer, Heidelberg (2008)
7. Gibbon, P., Frings, W., Dominiczak, S., Mohr, B.: Performance Analysis and Visualization of the N-Body Tree Code PEPC on Massively Parallel Computers. In: Proc. 11th ParCo Conf., Málaga, Spain, October 2006. NIC Series, vol. 33, pp. 367–374. John von Neumann Institute for Computing, Jülich (2006)
8. Labarta, J., Giménez, J., Martinez, E., González, P., Servat, H., Llort, G., Aguilar, X.: Scalability of Visualization and Tracing Tools. In: Proc. 11th ParCo Conf., Málaga, Spain, October 2006. NIC Series, vol. 33, pp. 869–876. John von Neumann Institute for Computing, Jülich (2006)
9. Brunst, H., Nagel, W.E.: Scalable Performance Analysis of Parallel Systems: Concepts and Experiences. In: Proc. 10th ParCo Conf., Dresden, Germany, September 2003, pp. 737–744 (2003)

Comparing the Usability of Performance Analysis Tools

Christian Iwainsky and Dieter an Mey

Center for Computing and Communication,
RWTH Aachen University
{iwainsky,anmey}@rz.rwth-aachen.de

Abstract. We take a look at the performance analysis tools Vampir, Scalasca, Sun Performance Analyzer and the Intel Trace Analyzer and Collector, which provide execution analysis of parallel programs for optimization and scaling purposes. We investigate, from a novice user's point of view, to what extent these tools support frequently used programming languages and constructs, discuss their performance impact and the insight these tools provide focusing on the instrumentation and program analysis. For this we analyzed codes currently used at the RWTH Aachen University: XNS, DROPS and HPL.

1 Introduction

High end computing machines for scientific applications are highly parallel computers, typically clusters of tightly connected shared memory parallel compute nodes with an increasing number of processor cores. The dominating paradigms for parallel programming in scientific computing are message passing with MPI, multi-threading using OpenMP or a combination of both. However development of parallel software is not an easy task.Tools have been developed to analyze and visualize a parallel program's runtime behavior and increase the users productivity in optimizing and tuning his code.

Performance analysis is typically performed in several phases. During the measurement phase, information about the runtime behavior is collected by instrumentation or sampling. Sampling an application means that at runtime the program is probed and a set of performance metrics is collected. This set usually contains the program counter, the call stack, memory usage and hardware-counters. For program tracing the program is instrumented beforehand to collect these metrics continuously as they occur [1]. In practice a developer has to carefully determine which method to use.

In the next step the data collected during the runtime measurement is analyzed, processed and finally presented to the developer for exploration.

Examples for such tools are the Intel tracing and analysis tools [2], the Sun Performance Analyzer [3], Vampir [4] and Scalasca [5]. These tools, though in continuous development, are already very sophisticated and their analyses are useful when parallelizing and optimizing applications.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 315–325, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In this work we investigate the usability of performance analysis tools and issues that a novice user might encounter. We compare these tools by means of benchmarks and real scientific applications and take a look at their performance impact to the applications at runtime. We also try to point out typical problems that a newcomer might run into. We focus on the runtime impact and therefore on the measurement phase, since the procurement of measurement data is a prerequisite for any analysis.

We recognize that this yields only a partial view of the usability, since we do not investigate in detail the exploration phase. Additionally, the exploration of the data is very subjective and difficult to rate as different users might have different preferences. However we feel, that the procurement of measurements is a crucial factor in the optimization of parallel programs, as measurements have to be performed iteratively after each program change to check for performance gains.

This paper is organized as follows: In section 2 we describe the tools under investigation and our test environment. In section 3 we look at the runtime performance impact on a few sample applications before we finally draw our conclusion.

2 Description of the Tools, Codes and the Test Environment

2.1 Tools

The performance analysis tools which we investigate in this paper are the Sun Performance Analyzer (Analyzer), Vampir Trace and Analyzer (Vampir), Scalasca and the Intel Trace Analyzer and Collector (ITAC).

The Analyzer is the only tool using the sampling approach that we investigated for this paper. The Analyzer can be applied to any program that runs on Linux and Solaris, provided it has been dynamically linked. As the target application is executed, the Analyzer interrupts the execution and collects desired metrics, like the program counter, the callstack and CPU counters. This information is stored in an internal buffer. Once this buffer is full, it is written to storage. Additionally, the Analyzer provides a special dynamic library that wraps the standard MPI library to obtain information about the MPI communication.

The ITAC, Vampir and Scalasca are instrumentation based tools, i.e. the tools have to preprocess the target program to insert measurement code. This instrumentation can be done at different levels, i.e. at the source-code level, at the binary level or at runtime. The gathering of MPI information is performed the same way as for the Analyzer by providing an MPI wrapper library. Vampir and ITAC perform only measurements at runtime, whereas Scalasca additionally performs the program analysis at the end of the program execution.

Vampir, Scalasca and the ITAC use function instrumentation mechanisms provided by modern compilers, which inserts hooks for call handlers at the beginning and the end of each function in the program. The tools then provide

Table 1. Overview of programs and used datasets

Code	Parallelization	Language	Lines of Code[1]	Source Files[1]
DROPS-OMP	OpenMP	C++	29.735	39
DROPS-MPI	MPI	C++	211.952	518
XNS	MPI	C and Fortran	47.866	102
HPL	MPI	C	35.163	150

a call handler which implements the measurement. Additionally, Vampir and Scalasca also provide an additional code instrumentation tool for OpenMP instrumentation [6].

Like the Analyzer, measurement data is initially stored in an internal buffer sharing the address space with the analyzed application. Once this buffer is filled up, it is written to disk interrupting the application.

All introduced tools can be influenced for better test results by environment variables or configuration files to modify default settings.

2.2 Codes Used for Testing

For this work we used the following codes, each representing a common programming and parallelization paradigm (see table 1):

1. DROPS[7] is an adaptive two phase CFD solver, written in C++ using modern programming language constructs like templates and classes. It is an example for a code using a very function-call intensive programming style with lots of nested function calls typical for C++. DROPS is available either with OpenMP [8] or MPI parallelization.
2. XNS[9] is a CFD solver based on finite elements written in Fortran and C. The code has been successfully adapted to several platforms. The version used for this work has been parallelized using MPI and is a representative for mature codes using common Fortran and C constructs.
3. The high performance computing linpack benchmark (HPL)[10] is a widely used benchmark for measuring system performance for the Top500 list (`www.top500.org`). It is written in C using BLAS libraries together with an MPI parallelization.

2.3 Test Environment

For our tests we used the Intel Xeon cluster of the RWTH Aachen University. This cluster consists of 266 nodes with 2 Quadcore Intel Xeon Harpertown CPUs running at 3GHz. Each node has 16GB of RAM, a DDR InfiniBand interconnect, local scratch storage as well as a system wide network file system. The operating system is Scientific Linux version 5.1. We compiled with version 10.1 of the 64 bit

[1] Data taken from complete source tree.

Intel compilers and linked to the Intel MPI library, version 3.1. Due to compiler issues we used the GNU compilers version 4.3 for the MPI Drops code. For HPL we used the MKL version 10.0 to provide the BLAS functionality.

The tests were performed with the Intel Trace Analyzer and Collector (version 7.1.0.1), Scalasca (version 1.0), Sun Analyzer for x86 (version 7.7) and Vampir Trace (version 5.4.4).

3 Application Performance

3.1 First Measurements

First we tested the tools in their basic, fully automated mode. This is how we assumed, that a newcomer might approach these tools. The data sets selected for these tests were chosen to be as small as possible with a maximum runtime of about 10 minutes while trying to maintain the typical behavior of the given application.

For Scalasca, Vampir and the ITAC we employed the recommended wrappers for automatic instrumentation and execution. For the DROPS-OMP sources we had to perform manual instrumentation of the header files since the automatic instrumentation does not process header files. It is recommended to apply Scalasca in two steps. We therefore measured all our test applications with the summary option first and then with the full trace. In oder to employ the Analyzer, we proceeded as described in the documentation as well.

We performed a complete set of measurements for each tool with each code with 8 MPI processes or 8 threads respectively on exclusive compute nodes. For these experiments the wall time was measured to capture possible setup and postprocessing overheads. We also set a limit of 100GB of measurement data per process/node as local scratch storage was limited. Measurements surfpassing this amount were aborted.

Both DROPS versions crashed during our first Scalasca profiling measurements. We investigated the resulting cores with a debugger and easily concluded that the instrumentation required a larger internal buffer to store the measurement data. Once we had increased these buffers the measurements completed. After these initial runs the instrumentation process terminated without a profile recommending an increase off buffer sizes. With larger buffers we obtained a correct and complete profile. At this point we already repeated the measurements for 3 times.

For DROPS-MPI we also encountered problems with Vampir. In this case the output of the instrumentation delivered a "Stack Underflow". We were not able to solve this problem with this Vampir Trace version. A later reinstrumentation with a newer VampirTrace 5.5 yielded correct measurements. However, this newer version has different settings and we had to reconfigure it to behave like the old version. Due to time constraints we were not able to repeat all measurements with the new version of Vampir.

The measurements of both DROPS codes with Scalasca in tracing mode was terminated as measurement had passed the 100 GB per process limit.

We also could not generate valid measurements with the ITAC in collect mode for both DROPS version and also for the HPL Code as the instrumentation system continuously noted a "file too large" message and finally crashed because of a bad file descriptor. We were unable to resolve this for this work.

Once we had a full initial set of runtime measurements, we observed, that for XNS and HPL the increase of runtime was acceptable, typically less than a factor of 7, comparing to the uninstrumented code (refer to table 2 for a complete set of runtimes). In contrast, both DROPS versions showed an enormous increase in runtime for trace based measurements with factors greater than 19. This occurred for Vampir and for Scalasca in both profile and trace mode.

For these first measurements we also list the runtimes as measured within the applications (compare table 3), which exclude tool specific pre- and postprocessing phases. These times show, that the applications are not only influenced by the pre- and postprocessing but also by the perturbation of the code through instrumentation. For HPL and XNS this perturbation is in the reasonable range of 1% to 453%. For the DROPS codes this factor for the traced based measurements is exrodinary high, with a factor of 27 to 868. The Analyzer performs in this case with an acceptable 11% to 519% runtime increase.

We also note the amount of measurement data written to storage, as large storage requirements can influence the usability of the tools (table 4). A brief evaluation showed that Scalasca,Vampir and the ITAC typically generated large amounts of data whereas the Analyzer was quite economical. Scalasca exhibits no consistent behavior for the XNS and HPL codes whereas Vampir generated little data for C and Fortran codes.

3.2 Refining the Collection of Runtime Performance Data

Once we had completed the initial measurements we investigated to what extend the tools could be tailored to the tested applications. We especially tried to reduce the runtime overhead and the amount of data gathered by the measurement tools.

In general, Scalasca, Vampir and the ITAC recommend that the user specifies a filter to remove uninteresting functions from the measurement process, by specifying a file with a list of function names that the analysis tool will ignore. To assist the user, Scalasca provides a utility that generates a list of functions from a previous gathered profile annotated with different kinds of information, like the number of function invocations, whether the function was on the path to an MPI call. By using this information we easily generated a filter that removed any function calls except those that eventually lead to MPI functions. For the OpenMP code we applied the same idea to the OpenMP constructs, filtering out all functions not leading to an OpenMP parallel region.

Vampir provides a different mechanism that computes a filter with the intention to reduce the measurement data to a given maximum percentage of the original trace. We used a aggressive target of 10% of the original measurement data for the first optimized measurement.

Table 2. Initial measurements, walltime

Code	Normal	ITAC	Scalasca		Analyzer	Vampir
			Profile	Trace		
DROPS-MPI	123 s	aborted	3974 s	canceled	475 s	2347 s
DROPS-OMP	75 s	aborted	24775 s	canceled	80 s	55059 s
HPL	777 s	aborted	784 s	988 s	1059 s	1123 s
XNS	68 s	336 s	98 s	430 s	327 s	120 s

Table 3. Initial measurements, timing from within applications

Code	Normal	ITAC	Scalasca		Analyzer	Vampir
			Profile	Trace		
DROPS-MPI	42 s	aborted	3687 s	canceled	218 s	1150 s
DROPS-OMP	63 s	aborted	24711 s	canceled	70 s	54711 s
HPL	751 s	aborted	751 s	766 s	1021 s	757 s
XNS	53 s	113 s	54 s	135 s	240 s	60 s

Table 4. Initial measurements, size of measurement data

Code	Normal	ITAC	Scalasca		Analyzer	Vampir
			Profile	Trace		
DROPS-MPI	-	aborted	16 MB	> 800 GB	276 MB	5.3 GB
DROPS-OMP	-	aborted	5.8 MB	> 100 GB	4 MB	2.4 GB
HPL	-	aborted	53 kB	299 MB	43 MB	105 MB
XNS	-	7.96 GB	550 kB	3.3 GB	320 MB	166 MB

Unfortunately the ITAC does not provide any such assistance for the definition of a filter. From our experience with the first measurements, we specified a very aggressive filter removing all but MPI communication functions from the trace.

In contrast to the tracing tools, the Analyzer does not have any filtering facility. Instead one can reduce the sampling frequency, delay the start time for measurement and define a data limit. We first decreased the sampling frequency from 10 ms to one second for the first optimization attempt.

With these filters and option changes we repeated the overview measurements with everything else remaining constant.

It should be noted that the filters differ between each tool yielding potential incomparable measurements. However, this sections aims to describe the approach of a novice user and the results should be interpreted accordingly.

Table 5 shows the results from this measurement set. For the sake of overview we omitted the measurements from the Scalasca profile run as the information is available in the tables 2 - 4.

We observed that the ITAC aborted again with file I/O errors for the HPL and the OpenMP version of DROPS. However DROPS-MPI successfully generated a trace this time. An analysis revealed that the trace barely contained any usable information at cost of a tremendous increase in runtime by a factor of 115. For

Table 5. First attempt of tool adaption

	Normal	ITAC		Scalasca Trace		Analyzer		Vampir	
	Time	Time	Data	Time	Data	Time	Data	Time	Data
DROPS-MPI	123 s	14233 s	401 kB	>7705 s	47 GB	267 s	276 MB	813 s	28 MB
DROPS-OMP	75 s	aborted		2201 s	1.6 GB	83 s	1.3 MB	42821 s	745 kB
HPL	777 s	aborted		784 s	299 MB	967 s	12 MB	1113 s	8.1 MB
XNS	68 s	156 s	32 MB	303 s	1.2 GB	294 s	313 MB	91 s	14 MB

the well behaved XNS a speedup was observed with almost no usable content in the trace.

With the filters we were now able to obtain a complete measurement for DROPS-MPI with Scalasca. Even though the analysis system ran out of memory during the analysis of the 47GB large tracefile we obtained a lower bound for runtime for tracing and analysis. A quick investigation, showed that a total of 232 GB of memory would be necessary to analyze the obtained trace as all trace data is loaded into memory. As with DROPS-MPI, the OpenMP version completed its measurement within the 100GB limit. Instead of taking 24775 s the runtime decreased to 2201 s with 1.4 GB of measurement data which the analysis system was able to handle. For XNS and HPL we observed some improvements in terms of runtime, however for XNS we lost information about computation functions as they did not end in MPI calls.

For the Analyzer we obtained a slight decrease in measurement data for all applications and some moderate gains in runtime. However, those gains were countered by a decrease of information for all codes as the individual runtime of functions was too short to be captured reliably. Further investigation showed that the traces consisted mostly of MPI information.

With the 10% filter Vampir Trace even surpassed Scalasca in runtime- and data improvements for the DROPS-MPI application. However this improvement came at the cost of trace data, which failed to provide any information about the application. An investiagtion of the filter showed that in the attempt to reduce trace size the filter-generator had marked allmost all functions for removal. For the OpenMP version of DROPS Vampir did improve in runtime, even though almost no trace data was written, resulting in an unusable trace. For the HPL and XNS codes some improvement in runtime was observed. The traces were still informative, although the details about functions not leading to MPI calls were unsatisfying for XNS.

3.3 Manual Optimization

As the first optimization step was unsatisfactory for XNS and the C++ based DROPS applications, we investigated the available information and manually specified custom tailored filters for each application. For this we tried to maintain as much information about the typical behavior of the program as possible. To reduce the substantial work overhead in this step and to increase comparability we designed a single filter for each test code and adapted this filter to all tools.

Table 6. Measurements for manual adaption attempt, total walltime

	Normal	ITAC		Scalasca Trace		Analyzer		Vampir	
	Time	Time	Data	Time	Data	Time	Data	Time	Data
DROPS-MPI	123 s	aborted		>3654 s	17 GB	199 s	91 MB	818 s	26 MB
DROPS-OMP	75 s	aborted		n.a.	>800 GB	91 s	4.3 MB	38013 s	2.1 GB
HPL	777 s	aborted		988 s	299 MB	1031 s	4.3 MB	1123 s	8.1 MB
XNS	68 s	209 s	37 MB	335 s	1.2 GB	176 s	76 MB	90 s	16 MB

For the specification of the DROPS filter we looked at the information available from previous analyses. The most obvious fact that we found was that the C++ code contained functions being called extremely frequently for each process. For example the [] operator of the vector class is called more than 1.6 million times per process. Several other functions were called approximately as often, many from the STL, others from the application, containing code targeted for optimization. However, to improve the measurement process we added the 10 most frequently called non STL functions to the filter list.

For DROPS-OMP we encountered a similar situation and again performed the same filter approach as for the MPI version.

For HPL we tried to further improve the fast results and filtered only the 3 most frequently called functions. When we looked at the XNS code and the profiles from previous runs, we learned that a large share of the runtime alteration and data footprint was caused by the measurement of the application's setup process. We designed therefore a filter to remove all setup functions from the measurement process. For the Analyzer we used the data limit and measurement start option to trigger a segment of analysis during the execution of the test codes. For this we used the default and smallest sampling intervall of 10 milliseconds.

With this setup and these filters we remeasured again (Table 6). With these changes and adapted filters we hoped that the ITAC would finally create usable information, however DROPS-MPI, DROPS-OMP and the HPL failed again with the already known file I/O error rendering the ITAC useless for those codes. For the XNS code we obtained a usable trace with enough information to optimize the application. For Scalasca we managed to decrease the trace size for DROPS-MPI whilst reducing the runtime again. However the analysis phase was still not able to process the 17GB of compressed trace data. For HPL and XNS the filters generated very useful measurement data which showed plenty of information necessary to start working on bottlenecks and load imbalances in the application.

For the Analyzer we observed an increase in performance. The information gathered in the measurement window showed the same characteristics as the information collected in the first run, outside this time window no information was obtained. The limitation to a certain time window and measurement size however helped to reduce the amount of data collected furthermore.

Table 7. All functions filtered

	Normal	ITAC		Scalasca Trace		Vampir	
	Time	Time	Data	Time	Data	Time	Data
DROPS-MPI	123 s	4338 s	32 kB	925 s	569 MB	913 s	6.6 MB
DROPS-OMP	75 s	aborted		2193 s	1.6 GB	42833 s	668 kB
HPL	777 s	1645 s	30 kB	988 s	299 MB	1109 s	88 kB
XNS	68 s	165 s	31 kB	283 s	1.2 GB	94 s	5.1 MB

Table 8. Measurements with PMPI interface only

	Normal	ITAC		Scalasca Trace		Vampir	
	Time	Time	Data	Time	Data	Time	Data
DROPS-MPI	123 s	138 s	126 MB	219 s	30 MB	135 s	24 MB
HPL	777 s	789 s	504 MB	784 s	87 kB	777 s	87 kB
XNS	68 s	373 s	24 MB	113 s	7.2 MB	103 s	7.1 MB

For Vampir the runtime of the instrumented applications yielded some ambiguous behavior. While the runtime hardly changed for the MPI applications compared to the previous measurement, the DROPS-OMP runtime improved by 11% generating 2.1 GB of measurement data. This data was somewhat informative as it showed much more detail about the applications. However, we were not able to obtain reliable information due to corruption of the application's runtime behaviour. For DROPS-MPI the filter was too aggressive showing only the main software components and the MPI runtime information. For HPL and XNS we were, as with Scalasca, able to obtain good results usable for performance analysis.

3.4 Limits of Tool Adaption

After both efforts of tailoring the tools to the codes, we investigated to what extend one can employ filters and setup parameters for the tools, to achieve the least impact on the target application. For this we specified filters that removed all information from the trace. From our point of view, the resulting measurement should then only be affected by the instrumentation overhead and the book-keeping of the file-output system. In addition to these new filters we also recompiled the applications without instrumentation and linked only the MPI tracing libraries of each tool to measure the overhead introduced by the MPI measurement system. Again we performed the same measurements as before. We omitted in this case the measurement with the Analyzer, as there was no way known to us to perform filtering or MPI-only measurements. Table 7 and 8 show the measurements obtained.

At this stage we were now able to confirm, that all tools were in principle capable of performing analysis for all codes, as we now obtained traces from all tools. We did not expect to obtain any noticeable amount of measurement data with those filters and were surprised that Scalasca still generated rather large

files for all codes (see table 7). A brief investigation showed, that even though we had specified MPI functions to be filtered, the Scalasca system ignored this and measured MPI functions regardlessly. As for all codes there was some noticeable overhead with these all-discarding filters. This overhead was low for XNS and HPL almost doubling the runtime. The instrumentation still had a large impact on the C++ codes. This is possibly explained by the tremendous amount of function calls.

As for the usage of the MPI wrappers we measured very minor perturbations of the applications' runtime and moderate amounts of data. With these traces we were able to conduct some basic performance analysis of the MPI communication. The information of the MPI communication alone might not be enough to conduct a complete performance analysis as the information is not sufficient to spot the causes of all communication problems and load imbalances.

4 Conclusion

We applied four different performance analysis tools to a set of four representative parallel application codes. Our measurements revealed that on the one hand the tools with proper configuration are quite suitable for analysis of C and Fortran codes like XNS or HPL, though obtaining such a configuration is not easy and requires a lot of practice for a newcomer. On the other hand our test displayed some serious problems when working with tracing tools and complex C++ codes as with both DROPS versions. In addition, we conclude that filtering is a viable option to adapt a performance analysis tool to an application. However, this tuning process can only start, once one has obtained a complete profile, which almost takes as long as a full trace. Additionally, this tuning process requires experience and foresight in order to utilize the full potential of the investigated tools, which a novice user might not have.

Future work will include an analysis of the difficulties with the DROPS codes and a scalability study of these tools on all four applications.

References

1. Shende, S.: Profiling and tracing in linux. In: Second Extreme Linux Workshop. USENIX, Monterey (June 1999)
2. Intel: Intel trace analyzer and collector 7.1, http://www.intel.com/cd/software/products/asmo-na/eng/306321.htm
3. Sun: Sun studio performance analyzer, http://developers.sun.com/sunstudio/overview/topics/analyzer_index.html
4. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The vampir performance analysis tool-set. In: Proceedings of the 2nd HLRS Parallel Tools Workshop, Stuttgart, Germany (July 2008)
5. Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Frings, W., Fürlinger, K., Geimer, M., Hermanns, M., Mohr, B., Moore, S., Pfeifer, M., Szebenyi, Z.: Usage of the scalasca toolset for scalable performance analysis of large-scale parallel applications. In: Proceedings of the 2nd HLRS Parallel Tools Workshop, Stuttgart, Germany (July 2008)

6. Mohr, B., Malony, A.D., Shende, S., Wolf, F.: Design and prototype of a performance tool interface for openmp. J. Supercomput. 23(1), 105–128 (2002)
7. Groß, S., Peters, J., Reichelt, V., Reusken, A.: The DROPS package for numerical simulations of incompressible flows using parallel adaptive multigrid techniques. Technical report, RWTH Aachen (2002)
8. Terboven, C., Spiegel, A., an Mey, D., Groß, S., Reichelt, V.: Parallelization of the C++ navier-stokes solver DROPS with OpenMP. In: ParCo, Malaga, Spain. John von Neumann Institute for Computing Series, vol. 33 (2005)
9. Behr, M., Arora, D., Benedict, N.A., O'Neill, J.J.: Intel compilers on linux clusters. Intel Developer Services online publication (October 2002)
10. Petitet, A., Whaley, R.C., Dongarra, J.J., Cleary, A.: Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers, http://icl.cs.utk.edu/hpl/

Real-Time Online Interactive Applications on the Grid (ROIA 2008)

Through the recent advancements in network technology, graphics cards and displays, a new type of application — real-time online interactive applications (ROIAs) — has become more and more popular. Everyday life has been affected and transformed not only by the use of Web technology, but also by collaborative multimedia applications, networked computer games, cooperative scientific visualizations, networked virtual environments and real-time graphical displays. Users and organizations dislocated all over the globe are enabled to work with a variety of tools and graphical user interfaces to communicate and jointly solve problems using sophisticated graphical interfaces. The grid and its technologies are known to provide a sophisticated basis for ROIAs and collaborative work.

This workshop at Euro-Par 2009 in Las Palmas de Gran Canaria comprised the best selected papers concerning the application and usage of real-time online gird applications, as well as the technologies supporting them in scientific and industrial contexts. The Workshop on Real-Time Online Interactive Applications on the Grid has offered possibilities to discuss the benefits of these applications for human users with a focus on up-to-date characteristics of hard-, soft and middleware aspects, to show the latest results, products or research prototypes to potential users, and to establish connections between developers and users of associated technologies. The attendants were asked to present and discuss the following topics:

- Interactive grid tools and environments
- Development of associated parallel and distributed computing solutions
- Integration of networking and grid computing technology
- Gaming approaches on the grid
- E-learning applications using grid technology
- Evaluation of existing ROIAs and practical experiences

Each of the submitted papers was reviewed by at least three international experts in this domain. The highest ranking contributions are presented here. We would like thank all colleagues and experts who helped in the reviewing process.

The problem of data collection and real-time processing of high-energy physics experiments is addressed by Bubak et al. in their contribution "Real-Time Performance Support for Complex Grid Applications". The authors present different monitoring systems from this area and the issues which arise by coupling theses systems.

Liška et al. present a framework designed for building real-time user-empowered collaborative environments to work primarily on high-speed networks with true high-bandwidth applications such as uncompressed high-definition video in their

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 327–328, 2009.
© Springer-Verlag Berlin Heidelberg 2009

contribution “CoUniverse: Framework for Building Self-Organizing Collaborative Environments Using Extreme-Bandwidth Media Applications”. They discuss the overall design of the system and describe in detail the prototype implementation.

The paper by Anthes et al. “Developing VR Applications for the Grid”, investigates the combination of virtual reality (VR) technology in combination with a grid infrastructure. A fast-paced VR entertainment application based on the inVRs framework is ported on the edutain@grid middleware, by exchanging its network component.

Another contribution from the edutain@grid project, “An Information System for Real-Time Online Interactive Applications”, by Nae et al. introduces a novel information system that provides support for ROIA deployment and monitoring. A variety of experiments are conducted which investigate the performance and scalability of the system.

In “Securing Real-Time Online Interactive Applications in edutain@grid”, Ferris et al. present analysis, design and implementation of security facilities within the edutain@grid infrastructure. They describe their solutions to security issues on the different layers on the infrastructure.

The contribution by Landersthamer et al., “The edutain@grid Portals — Providing User Interfaces for Different Kinds of Actors”, investigates the use of user interfaces in order to interact with edutain@grid. Instead of relying on Web portals only, they present approaches to connect with a C++ API to an edutain@grid ROIA.

In the paper “Using RTF for Developing Multi-Player, Online” by Ploss et al., the creation of ROIAs by using a real-time framework in order to allow for scalability by keeping up interactivity is demonstrated. Code examples illustrate their approach and describe the development of such ROIAs in a detailed way.

Christoph Anthes
Thomas Fahringer
Dieter Kranzlmüller

Real-Time Performance Support for Complex Grid Applications

Marian Bubak[1,2], Włodzimierz Funika[1], Bartosz Baliś[1],
Tomasz Szepieniec[2], Krzysztof Guzy[2], and Roland Wismüller[3]

[1] Institute of Computer Science AGH, al. Mickiewicza 30, 30-059 Krakow, Poland
[2] ACC CYFRONET AGH, Nawojki 11, 30-950 Krakow, Poland
[3] Universität Siegen, Hölderlinstr. 3, 57068 Siegen, Germany
{bubak,funika,balis}@uci.agh.edu.pl,
t.szepieniec@cyfronet.edu.pl, kjguzy@gmail.com,
roland.wismueller@uni-siegen.de

Abstract. One of the tasks in the EU IST `int.eu.grid` project is to create a pilot application which exploits Grid computational resources, based on the HEP application coming from the ATLAS project. When bringing the HEP application to the Grid, one of the problems is how to distribute the computations in an effective way to make the most of available Grid resources. Some support from monitoring is needed to assist in an optimal computation distribution. The HEP application is supported with monitoring information coming from two systems: JIMS and OCM-G. In this paper we present the issues of providing data for Real Time Dispatcher, based on the functionality of the two monitoring systems systems coupled into a single infrastructure. It is aimed to support fulfilling the real-time requirements posed by the HEP application.

Keywords: grid, monitoring, HEP, LHC, application optimization, MBeans.

1 Introduction

The objective of the EU IST Interactive European Grid (int.eu.grid) project [1] is to deploy an advanced Grid empowered infrastructure in the European Research Area (ERA). Within the project, research on its pilot application from the domain of High Energy Physics (HEP application) [4] is aimed at using the interactive grid environment to support processing of data coming from the biggest particle accelerator - the Large Hadron Collider (LHC). The current architecture of the High-Level Trigger (HLT), part of this system, requires building a massive computer farm of order of 1000s processors employed just for the selection of the most interesting collisions; it heavily involves the network infrastructure. So far in the local processing model exploited in the ATLAS TDAQ system, the data on the events were transferred for pre-processing to local computing farms. The idea behind the HEP application gridification is to delegate part of pre-processing tasks to computer resources available on the Grid.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 329–338, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The High Energy Physics application (HEP) poses hard challenges related to the Grid environment, specifically, in case of LHC experiments, their requirements go far beyond the ones typically involving storage and computational issues. The main task of the ATLAS TDAQ system is to select interesting events out of 1 Giga interactions per second generated in collisions at the LHC accelerator and record them on permanent storage with frequency of O(300 Hz). The system is based on two levels of online selection algorithms. The TDAQ system is logically divided into a fast First Level Trigger (L1), and High Level Trigger system (L2) which involves the next two selection stages. The First Level Trigger (L1) provides an initial reduction of the LHC's 40 MHz nominal bunch crossing rate to 75 kHz. Given a mean ATLAS event size of 1.6 Mbyte, this corresponds to a throughput of ca. 120 GB/s. The first stage of L2 reduces the event rate further to 3.5 kHz, which corresponds to a total throughput of about 6 GByte/s out of the event building system. The data fragments of events accepted by the first stage of L2 algorithms are collected by the event building nodes (SFIs) (see Fig. 1) from detector buffers. The resulting complete event fragments are then sent to the Event Filter Processors (EFPs) (see Fig. 1) for the last selection stage (i.e. the second stage of L2).

To enable the filtering of events during the second stage of L2, a need of 1600 EFP nodes (dual-CPU machines) is foreseen. To meet this challenge, it is proposed to extend the EFP system by the possibilities of a Grid infrastructure to allocate necessary processing power with fast access to get the data processed in realtime (O(1s)). A non-timely response to an event leads to the irreversible loss of the data coming from the real LHC experiment. Taking into account that submission of jobs to the Grid may take quite an amount of time, the use of the pilot jobs idea [2,3] is a proper solution. Just before the experiment starts, jobs are submitted to the Grid. These jobs allocate resources and establish a communication channel. They start waiting for tasks obtained from the experiment.

To use this idea, some middleware support for the HEP application is needed. The Real-Time Dispatcher (RTD) is a software solution dedicated to facilitate the delegation of computations to the Grid infrastructure. It coordinates the distribution of data obtained from the ATLAS TDAQ system to remote processing tasks (PT). A remote processing task is a grid job which is submitted in advance to wait for data to work on. The data is sent by a *proxy processing task* (proxy PT) which is running on local resources in CERN. The proxy PT is behaving in the same manner like an ordinary local processing task with the exception of delegating an actual computation to a remote PT. When remote PTs obtain the data from the proxy they perform the computations and return the result which is further used by the ATLAS TDAQ system to judge whether this data could describe some interesting collisions from the scientific viewpoint. On having returned a result the remote PT is ready for getting the next data to process. So the main RTD's purpose is to bind the proxy and a remote PT. When the proxy PT starts, it requests a remote PT to be assigned. As RTD has a pool of remote PTs, it must decide which one will be connected with the requesting proxy PT. The selected remote PT should process the coming events' data at

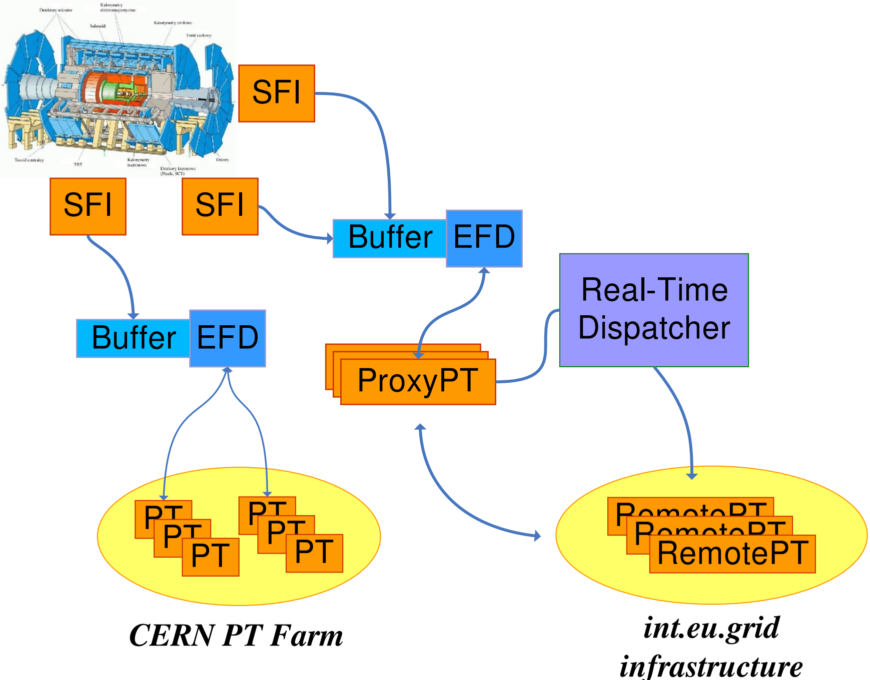


Fig. 1. The idea of HEP application

the fastest rate. Thus to make proper decisions, the RTD needs to be provided with needed information by a monitoring facility. For this goal we exploited two monitoring facilities, one for obtaining data on network load, while for better performance and less resource usage (mainly, main memory) we decided to use another lightweight monitoring facility which provides fast access to the data regarding the processing performance of the nodes where remote PTs run.

Fig. 1 presents a general concept of the HEP application. One can see how the local processing (on CERN PT Farm) is augmented by the use of Grid resources (based on the int.eu.grid's infrastructure) with RTD, proxy, and remote PTs.

In the following sections, we focus on the requirements for the monitoring support from the HEP application and make a brief overview of the monitoring systems we used to fulfill these requirements. Then the issues of coupling these systems are discussed. We will give details of co-operation of these facilities over the application life-time to match the real-time application requirements. The last section includes conclusions and future work.

2 Monitoring Requirements for the HEP Application

As mentioned above, RTD needs some monitoring information to choose a remote PT which should process the events' data at the fastest rate. To make this

choice, the RTD is to be provided with the information on the network load and resources usage of the worker nodes where remote PTs are running. Information on the load of a network link is important to determine whether a given grid site is able to receive the data fast enough. On the other side, knowing the resource usage of worker nodes (CPU load and the amount of available memory), we can distribute computations to less loaded nodes. Making use of this information, RTD can choose the most appropriate remote PT at the moment. To fulfill the monitoring requirements from the HEP application we decided to use two monitoring systems: JIMS and OCM-G. These monitoring systems will be described concisely in the following subsections before we proceed to the discussion on an integrated monitoring infrastructure for the HEP application.

2.1 JIMS Monitoring Infrastructure

JIMS – the JMX-based Infrastructure Monitoring System [5] is based on the Java Management Extensions platform (JMX) where monitored resources are represented as MBeans (Managed Beans), simple Java objects, installed on MBean Servers. JIMS can automatically adapt itself to operating systems, kernels, and IP protocols. It provides an auto-configuration facility (cluster and Grid level auto-configuration) and dynamic deployment of proper monitoring sensors. All these features make it well suited for Grid environments.

Figure 2 presents the JIMS architecture. It consists of three types of agents. The role an agent performs depends on the modules where the agent is deployed. One agent provides the global discovery service which delivers an integration layer, it finds all running SOAP Gateway agents, thus provides an overall view of the Grid. The proxy agent, in turn, finds and provides access to all currently running monitoring agents on a cluster. An ordinary monitoring agent, which is running on a worker node, provides monitoring information collected by its

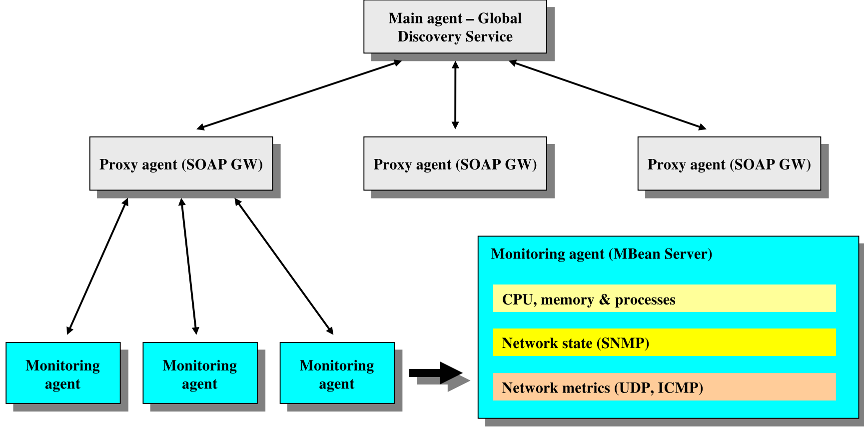


Fig. 2. JIMS components

sensor modules, i.e. MBeans. The figure gives a few examples of the information the sensors can provide.

2.2 OCM-G Monitoring System

OCM-G – OMIS-Compliant Monitoring system for the Grid [6,7] is a system for on-line monitoring of interactive Grid applications. It supports parallel/ distributed applications running across multiple sites. Due to the efficient request-reply mode of operation, OCM-G can underly development-support tools, specifically, performance analysis tools, like the GPM [8] performance measurement tool. It provides special support for monitoring applications exploiting message-passing and other programming paradigms. OCM-G can also provide information on the nodes where the application processes are running.

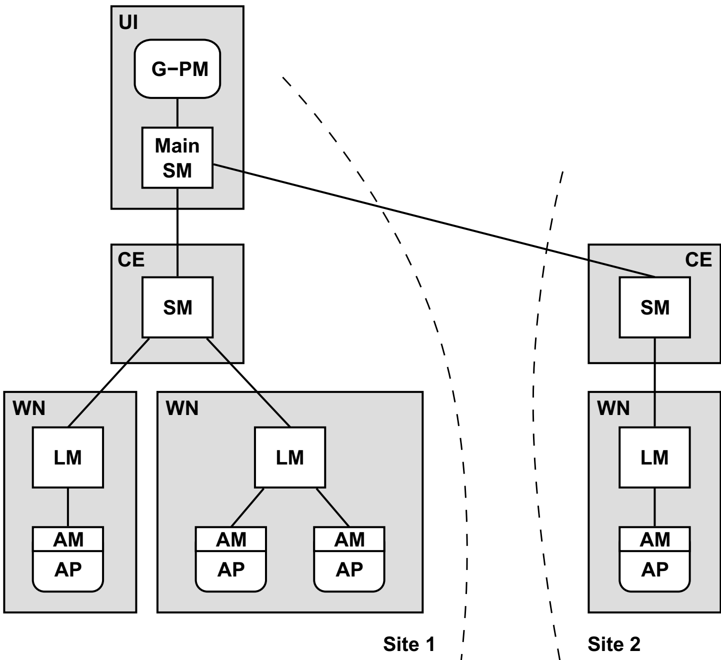


Fig. 3. OCM-G components distributed on the Grid

As shown in Figure 3, the OCM-G system comprises three types of monitor modules. *Main Service Manager* (MainSM) is a central component, one per each user. It is an access point to the system for a user. The location of MainSM is configurable. *Service Managers* (SM) are located in each site of the Grid, typically, on Computing Element machines (CE). *Local Monitors* (LM) are located on each Worker Node (WN) where the application processes (AP) under monitoring run and they actually perform the monitoring tasks. *Application Modules* (AM) are parts of OCM-G that are linked to the application processors.

2.3 Monitoring Strategy for the HEP Application

As the main monitoring data provider for RTD, the JIMS system has been chosen. JIMS is meant to be deployed in each site of the Virtual Organization, intended to support the HEP application (HEP VO). HEP VO is responsible for providing a necessary environment to run the HEP application on the Grid. Available grid resources are carefully checked during a certification procedure. Next, SLA [10] is signed and resources are included into a HEP VO. The HEP VO is endowed with a portal which simplifies the usage and management of VO.

JIMS will measure the available bandwidth to sites where events' data are processed. For this purpose, only one JIMS agent is needed, probably, on a computing element. In addition to the network load, RTD needs to obtain some data about worker nodes status: CPU load and memory usage, very frequently and rapidly. JIMS could provide this information but to do this it should be installed also on every worker node of HEP VO. To save resources of worker nodes (mainly, main memory) we decided to take the advantages of the OCM-G monitoring system, instead. For collecting monitoring information, OCM-G uses a monitor which is much more lightweight than a JIMS agent as to the memory required for its operation. Moreover, on a worker node the OCM-G monitor is started together with an application process, thus, the monitoring system don't exploit resources of the worker nodes where the application in not running at the moment. For this reasons we decided to monitor WNs using OCM-G which is intended to provide monitoring information to a JIMS agent. Next, RTD will obtain this information from JIMS. In the next section, we will give some details of the integration of OCM-G into JIMS .

3 Integration Mechanism

To integrate OCM-G with JIMS we needed to implement a new sensor module for JIMS. As sensor modules JIMS exploits MBeans. An MBean runs in a JIMS monitoring agent (MBean server) and provides some monitoring functionality. Thus, we created an MBean which hides the OCM-G system and provides the data about nodes' CPU load and memory usage through its exposed interface.

When loaded and started by a JIMS agent, the MBean spawns OCM-G's Main Service Manager process and connects to it. Then the Main Service Manager is waiting for Local Monitors to connect to (a Local Monitor is actually connected to Main Service Manager using a Site Service Manager). A Local Monitor is started whenever a remote PT is launched on a worker node. This is done by a script which also stops the Local Monitor when the remote PT exits. Local Monitors find the Main Service Manager's contact by reading a configuration file with host and port information. The list of currently monitored worker nodes can be obtained by reading the `NodeList` MBean's attribute. The value of this attribute is a comma separated list of hosts' names.

The MBean provides the user with the operations to start (`start`) and stop (`stop`) the underlying OCM-G system (Main Service Manager process). But these operation are rather not used directly as they are invoked by the MBean

when JIMS agent starts and stops. For the client, the interesting operations are those which provide needed monitoring information. At the moment, there are six operations provided by the MBean. All these operations take a host name as an argument. The first three ones: `getNodeLoadAvg1`, `getNodeLoadAvg5`, `getNodeLoadAvg15` return the load average values for a given worker node (i.e. the number of jobs in the run queue or waiting for disk I/O averaged, respectively, over 1, 5, and 15 minutes). They are the same as the load average values returned by *uptime* and other programs. The next operation, `getNodeMemFree`, returns the amount of available host's main memory. Invoking the last operation, `getNodeSwapFree`, we can get the amount of remaining swap space. When one of the MBean's operations is called, the MBean constructs a proper request and submits it to the OCM-G system. When an OCM-G's reply arrives, it is processed and returned within an MBean's operation result.

OCM-G is installed on HEP VO sites as an experiment software [9]. Therefore, the VO software manager is responsible for installation, configuration, and

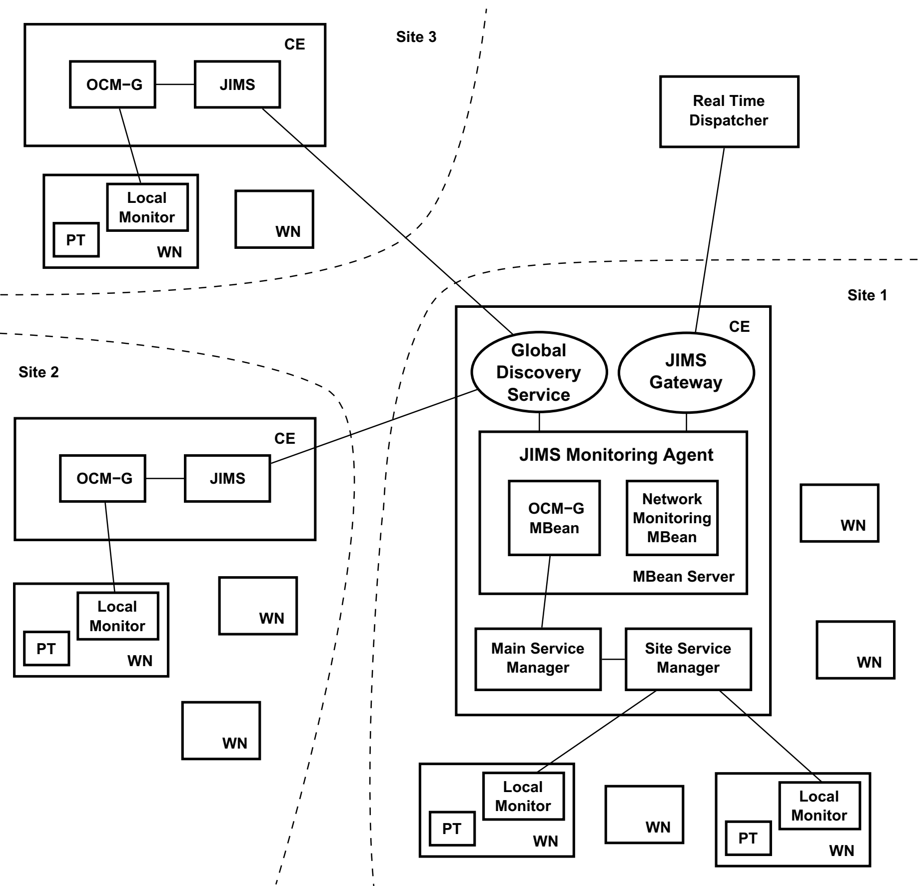


Fig. 4. Overview of JIMS and OCM-G integration

updating the software. The advantage is that a contact with a site administrator for these activities is not needed. JIMS agent is installed on a single host in the site (possibly, CE) and runs as a service. This agent has our MBean deployed and is an access point to the monitoring information collected by OCM-G's Local Monitors running on site's WNs. Using the JIMS infrastructure the data from all sites can be supplied to RTD. In order to facilitate a client's interaction with JIMS, there has been added a possibility to access all discovered agents using a single connection. Earlier, to find the address of an agent, a client had to connect to the Discovery Service first. Then, when the client obtained the address, it could connect to an agent directly. Now, all agents provide the Discovery Service and have an ability to forward a client request to other discovered agents. Thus, using a single connection with a chosen agent, a client can also interact with other agents. This makes easier gathering monitoring information from all the sites involved in the HEP application.

Figure 4 presents an overview of OCM-G and JIMS components and relationships between them. In one site its components are shown in details while in the two remaining ones the details are hidden. The RTD is connected to a JIMS agent. This agent serves as a JIMS Gateway for the site so it is accessible from outside. The agent also provides Discovery Service and owing to this the RTD can access other JIMS agents in the other sites. Within the depicted JIMS agent, there can be seen two MBeans. Our MBean is connected with the OCM-G system and there are OCM-G's monitors on the worker nodes where PTs are running.

The information provided by the OCM-G MBean can be accessed in different ways, e.g. through e.g. JIMS GUI. In Figure 5 a GUI, called JIMS Manager is

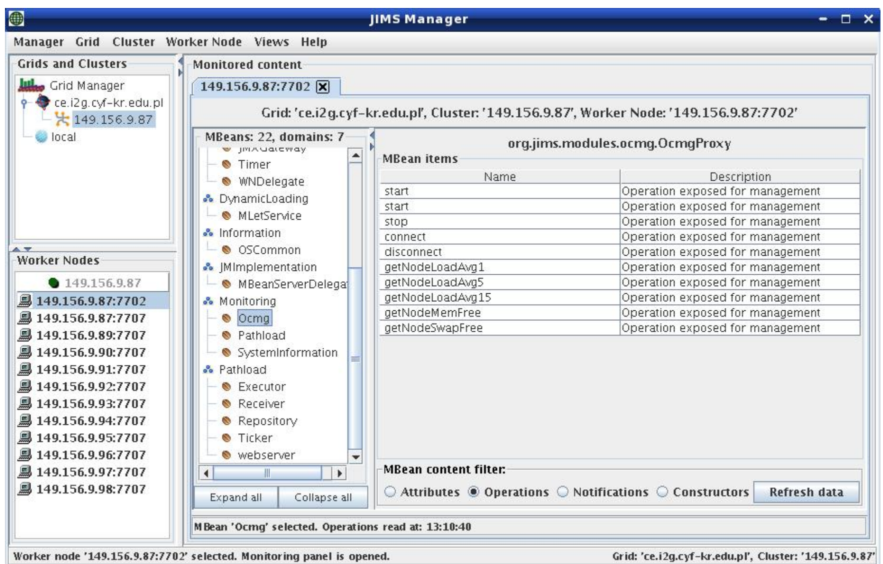


Fig. 5. OCM-G MBean in the JIMS Manager window

shown. On the left top panel (*Grids and Clusters*) there is a list of known JIMS agents which were specified in a configuration file. With selecting one of these agents we can start a process of finding other ones that will be shown on the left bottom panel (*Worker Nodes*). In the figure, on the left, there is an open tab for a one of the discovered agents. A list of MBeans the agent runs is presented. Once the OCM-G's MBean is selected, on the right there are shown our MBean's operations.

4 Conclusion and Future Work

The solution for middleware support, in particular, this related to monitoring functionality, presented in this paper was designed for the HEP event processing problem, which poses some rigorous time requirements for data processing. Usually, large delays in the event processing end up in the overloading of the queues holding the data coming from the trigger.

In this paper we focused on the requirements for monitoring support for Real Time Dispatcher in choosing proper remote Processing Tasks for performing computations within the High Energy Physics application explored as a pilot application in the `int.eu.grid` project. The designed monitoring strategy and the idea of how we had integrated OCM-G monitoring system into the JIMS infrastructure for the monitoring of the HEP application were shown.

At the moment, a working version of our solution is tested with RTD and remote PTs. In the future we intend to improve the reliability of OCM-G by an elimination of Main Service Manager component. This component is important when OCM-G is used independently on the Grid. It gathers monitoring information coming form all Site Service Managers (one per site). In our scenario, we've got one OCM-G system per site so the Main Service Manager is not needed and the Site Service Manager should be accessed by a client (OCM-G MBean) directly. Our further focus will be on the comparative study of the functionality of the monitoring infrastructure under discussion and similar systems, which will involve experiments with MonALISA [11], a distributed, auto-configuring and auto-deploying monitoring system that can provide quasi-real-time information about huge number of nodes. This will also enable us to study the scalability of of our monitoring system.

Acknowledgements. We are very grateful to dr. Renata Slota for valuable discussions. This research is partly supported by the EU IST `int.eu.grid` project IST-031857, the corresponding SPUB-M and the EU IST CoreGRID project.

References

1. EU IST int.eu.grid project's web page, `http://www.interactive-grid.eu/`
2. Thain, D., Livny, M.: Building Reliable Clients and Services. In: Foster, I., Kesselman, C. (eds.) The Grid: Blueprint for a New Computing Infrastructure, pp. 297–299. Morgan Kaufmann, San Francisco (2004)

3. Globus Falkon framework page, http://dev.globus.org/wiki/Incubator/Falkon
4. Dutka, L., Korcyl, K., Zielinski, K., Kitowski, J., Slota, R., Funika, W., Balos, K., Skital, L., Kryza, B.: Interactive European Grid Environment for HEP Application with Real Time Requirements. In: Bubak, M., Turala, M., Wiatr, K. (eds.) Proceedings of Cracow 2006 Grid Workshop, Cracow, Poland, October 15-18, 2006, pp. 11–20. ACC Cyfronet AGH (2007)
5. Zielinski, K., Jarzab, M., Wieczorek, D., Balos, K.: JIMS Extensions for Resource Monitoring and Management of Solaris 10. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 1039–1046. Springer, Heidelberg (2006)
6. Balis, B., Bubak, M., Funika, W., Wismueller, R., Radecki, M., Szepieniec, T., Arodz, T., Kurdziel, M.: Grid Environment for On-line Application Monitoring and Performance Analysis. Scientific Programming 12(4), 239–251 (2004)
7. OCM-G project home page, http://grid.cyfronet.pl/ocmg/
8. Wismüller, R., Bubak, M., Funika, W.: High-level application specific performance analysis using the G-PM tool. In: Di Martino, B., Kranzlmüller, D., Dongarra, J. (eds.) EuroPVM/MPI 2005. LNCS, vol. 3666, pp. 317–324. Springer, Heidelberg (2005)
9. Experiment Software Installation, https://edms.cern.ch/file/498080/1.0/SoftwareInstallation.pdf
10. Skital, L., Janusz, M., Slota, R., Kitowski, J.: Service Level Agreement metrics for real-time applications on the Grid. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 798–806. Springer, Heidelberg (2008)
11. The MonALISA web site, http://monalisa.cern.ch/monalisa.html

CoUniverse: Framework for Building Self-organizing Collaborative Environments Using Extreme-Bandwidth Media Applications

Miloš Liška and Petr Holub

Laboratory of Advanced Networking Technologies
Faculty of Informatics
Masaryk University
Botanická 68a, 621 00, Brno
Czech Republic
xliska@fi.muni.cz,
hopet@ics.muni.cz

Abstract. In this paper, we present a framework called CoUniverse, designed for building real-time user-empowered collaborative environments to work primarily on high-speed networks with true high-bandwidth applications such as uncompressed high-definition video. The system is designed for unreliable experimental infrastructures and therefore its operation relies heavily on self-organizing principles—this is also useful approach for extending it to larger infrastructures. When media stream bitrate is comparable to a capacity of the links, the additive assumption no longer holds and the system needs to have a sophisticated scheduling. The scheduler is conceived as a flexible plug-in for the CoUniverse framework. In this paper, we present a formal scheduling model based on constraint programming including evaluation of its prototype implementation. CoUniverse is designed to utilize external media applications, so that a wide variety of existing tools can be used. The whole system has been prototyped and demonstrated, e. g., during international demonstration on the GLIF 2007 workshop.

1 Introduction

The Grid environment is nowadays understood not only as a manner how to share computational resources or data storage facilities but may be understood in a more general way as an infrastructure for sharing of various types of capacities and for virtual collaboration. In this context it also includes high-quality collaborative environment. High-quality collaborative environment must be able not only to transmit media streams with the best possible quality, but also it has to be capable of accommodating changes in the underlying infrastructure. While multipoint transmissions of low-latency uncompressed high-definition media streams at 1.5 Gbps provide the desired quality, they have very high demands and lack adaptivity to changing networking conditions. Bitrate of such media streams becomes comparable even to the current highest-speed network links

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 339–351, 2009.
© Springer-Verlag Berlin Heidelberg 2009

(10 GbE or OC-192) and thus scheduling of media streams to network links needs to be done carefully. Furthermore such an environment comprises large number of components which can become very hard to orchestrate manually. Manual orchestration of components makes virtually impossible reacting to network events in time short enough to minimize impact of events on the users' experience.

In this paper, we propose a self-organizing collaborative environment framework for real-time network transmissions called CoUniverse. CoUniverse is designed as an application middleware capable of orchestrating collaborative environments like the one described above. Careful separation of control plane from data plane within CoUniverse allows for optimization of these two networks for different purposes. For the control plane, CoUniverse framework uses peer-to-peer (P2P) network communication substrate which adds necessary robustness and reliability even on experimental infrastructures. We have designed CoUniverse as self-organizing system capable of automated user-empowered steering and encapsulation of legacy media applications (i.e., third-party components of the collaborative environment which are completely unaware of the middleware). Our framework is also capable of responding to changes and outages of the underlying Grid (network and processing) infrastructure. CoUniverse introduces a concept of pluggable scheduler to address the self-organization aspect of the framework by the means of planning the transmissions of the media streams over particular network links and creating corresponding configuration for respective collaborative environment components. Not only that the streams with constant parameters can be configured to individual links, but strategies for using alternative streams and/or adjusting stream parameters may be defined. This allows for using, e.g., 250Mbps compressed stream instead of 1.5Gbps uncompressed when links required for 1.5Gbps stream are not available.

This paper is further structured as follows. Basic design principles used for proposing architecture of CoUniverse are discussed in Section 2. Resulting proposed architecture including overview of basic components and organization of the network is described in Section 3. The system has been prototyped including preliminary version of the scheduler as discussed in Section 4. The system has already been demonstrated during several events and its evaluation especially with focus on performance of current version of the scheduler is given in Section 5. Because the field of collaborative environments is rapidly moving forward, we brief related work in Section 6. The paper is concluded by tackling future research tasks and proposing further applications for CoUniverse in Section 7.

2 Design Principles

The CoUniverse is organized as one or more *collaborative Universes*, where the actual collaboration takes place, and a *Multiverse*, used for registration and lookup of clients and Universes. The collaborative Universes are intended to accommodate collaborative groups of limited sizes [1] and thus can implement functionality that may be hard or impossible to deploy at large. This includes

features like sophisticated scheduling and aggressive monitoring of components and network that provides basis for fast reaction to problems that may occur. On the contrary the Multiverse provides a very limited functionality, it has to scale well with respect to large number of participating nodes.

In terms of self-organization, CoUniverse is capable of reacting to events in the system, namely to events raised by users, nodes, and by the monitoring. It includes applications being started/terminated, network links being turned up/down, changes in link parameters (capacity, loss, latency, jitter), nodes being added to and removed from the Universe and nodes being reconfigured.

CoUniverse needs to have a scheduler to support applications with media streams comparable to network link capacity. The scheduling objectives may vary: for simple interactive applications with fixed quality, it usually includes minimization of media distribution latency and possibly minimization of number of nodes involved in the network. For more complex applications where quality is an adjustable parameter, maximization of the quality may also be included. Output of the scheduler has to include not only the plan itself, but also a *workflow* describing how to implement the plan, as there are many functional dependencies. For instance, network links need to be allocated prior to starting media applications that will send data over them.

Because the CoUniverse is designed to integrate high-bandwidth applications, it is necessary to interface with services provided by advanced networks like lambda services [2] or network resource allocators [3].

The whole system follows the user-empowered paradigm [4,5] as much as possible. The CoUniverse doesn't require administrative privileges especially over the network and components which means that the system is able to run entirely in user space.

3 Proposed Architecture

3.1 Network Organization

As discussed above, the CoUniverse is organized as one or more collaborative Universes and a Multiverse. From the networking point of view each Universe consists of a *control plane* used for control communication of all components of the Universe and one or more *data planes* used for actual data exchange between Universe components. Both control plane and data planes are forming an overlay networks on top of an actual physical network infrastructure.

The Multiverse and the control planes of collaborative Universes are based on a P2P networking substrate which provides necessary robustness for the Multiverse and the control planes. Moreover a P2P substrate provides functions like clients and Universes description, naming and addressing, lookups and reliable data transfers.

The data planes of the collaborative Universes are based on available physical networking infrastructure. The data planes are optimized for maximum performance and minimum latency when transmitting data between the components

of the Universe. As data planes are virtual overlays over a physical networking substrate, they exist only in case when there is an Application Group (see bellow) to utilize it. The system is designed with user-empowered paradigm in mind and thus it naturally relies on using application-level media "routers" and distributors (reflectors, Active Elements [5]) for multipoint data distribution.

3.2 Collaborative Universe

Collaborative Universes, as shown in Figure 1, consist of nodes, each of which runs Universe Peer client. Universe peers are providing a base for communication among the Universe components, managing underlying node configuration and steering media applications configured on the very node. Nodes within the Universe are aggregated into *network sites*, usually representing all nodes of a single site participating in the collaborative Universe. To give more precise definition, a network site is a set of collocated nodes, where each site may have one or more users participating. Expressed using terminology defined below, typical property of all nodes within one site is that there are no consumers consuming data from producers from the same site (this definition doesn't include media distributors).

Each network node is configured by specifying *(i)* a list of its physical network interfaces and their parameters, *(ii)* a list of Media Applications which are installed on the node, and *(iii)* a network site the node belongs to. A Media Application is any application which is used to create the collaborative environment and which produces or consumes a media stream (e. g., videoconferencing clients, audioconferencing clients, data distributing Active Elements (AE) [6], etc.). All Media Application producers (except AEs) are producing exactly one media stream which is then sent to exactly one consumer.

Media applications are organized into Application Groups (AG). AGs are then generalizing a particular functionality of the collaborative environment

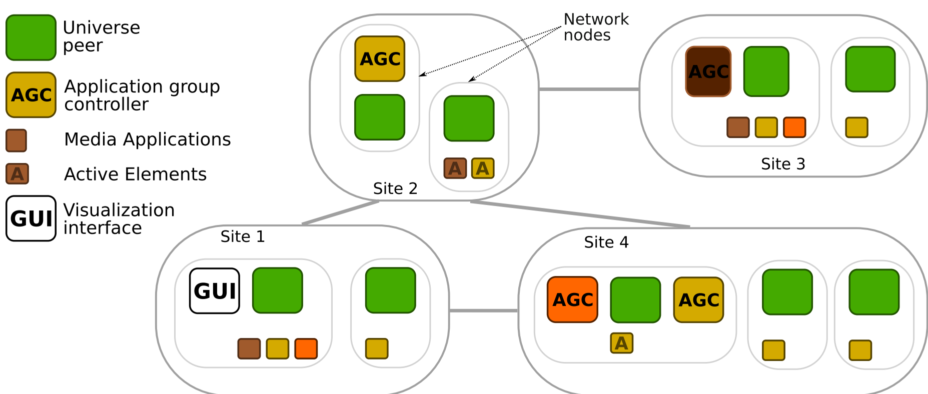


Fig. 1. Scheme of the Collaborative Universe with its components. Different colors for Media Application squares mean different media applications.

(e. g., audio or video conferencing, desktop sharing etc.). Media applications within an AG are orchestrated using an Application Group Controller (AGC). AGC is a service running on top of at least one of the regular Universe peers.

The purpose of the AGC is to collect node configurations from all peers within the collaborative universe, assemble a topology of universe data planes, invoke a scheduler to schedule the media streams of corresponding media applications to a physical network links, create a configuration for each media application based on scheduled media streams and finally send the configuration together with data plane topology to respective universe peer. The universe peer in charge then adjusts the configuration of steered media application so that it corresponds to respective scheduled media stream. The scheduler within the AGC is invoked either manually (especially for the first time) or automatically as a reaction to a change Collaborative Universe state (e. g., new node appeared, a node is not reachable using a particular network link etc.).

3.3 Monitoring

Each Universe peer comprises monitoring of steered media applications, network links of a physical networking substrate which might be used to build the data plane for the media applications and the network links that are actually part of some data plane. Monitoring of the data planes network links is more aggressive than monitoring of network links of generally available physical networking substrate since the links of data planes are actually used for media applications data exchange. At the same time, the links that are not used in any of the Universe data planes need to be monitored less frequently just so that the AGC eventually has a notion of their state when some event in the Universe occurs and those links might be used for some newly scheduled media streams.

3.4 Visualization

Visualisation gives an overview of an actual collaborative Universe state to the user. Our goal is to provide a dynamic visualisation displaying on one hand topology of the physical network between nodes of the collaborative Universe, which might be used to build the data planes, and on the other hand active (currently scheduled) media streams. Visualisation of active media streams is extremely useful especially when incorporating data from network and applications monitoring. Moreover, users can also easily find out whether the schedule chosen for a given network topology has the desired effect (i.e., users can see, talk to, or collaborate with each other in the way it was intended in a particular collaborative universe).

3.5 Scheduling Network Model

In order to describe the scheduling algorithms implemented into the CoUniverse framework, we need to introduce formal notation first. In this section, only the notation is described, while the actual constraints used for scheduling are available in Section 4.

Let I be a set of all network interfaces, $i \in I$ a network interface. Furthermore let N be a set of all nodes in the Universe, $n \in N$ a particular node. Then $\mathrm{node}(i) = n$ where $n \in N$ is a node n with configured network interface i.

Let $l = (i, j)$ be a network link for $i, j \in I$. Then $L = I \times I$ denotes a set of all network links and we denote a particular network link as $l \in L$. We can define following properties of a network link l: $\mathrm{begin}(l) = i$ such that $l = (i, j) \wedge i, j \in I$ is the originating interface i of the link l, $\mathrm{end}(l) = j$ such that $l = (i, j) \wedge i, j \in I$ is the ending interface j of the link l. $\mathrm{cap}(l)$ denotes the link capacity.

Finally, let P be a set of producers where $p \in P$ is a media application producer, C a set of consumers where $c \in C$ is a media application consumer and M set of media distributors where $m \in M$ is an Active Element (AE). Producers, consumers and media distributors are running on the nodes $n \in N$. Let $\mathrm{consumers}(p)$ where $p \in P$ be a set of consumers for a particular producer p. Thus $\bigcup_p \mathrm{consumers}(p)$ is a set of all active consumers, i.e., those that have requested a data stream from some producer. In the opposite direction, $\mathrm{producer}(c)$ is the requested producer for the consumer c. Furthermore we define $\mathrm{node}(p) = n$ where $n \in N \wedge p \in P$, $\mathrm{node}(c) = n$ where $n \in N \wedge c \in C$ and $\mathrm{node}(m) = n$ where $n \in N \wedge m \in M$ as a parent nodes of the producer p, the consumer c and the media distributor m. A media application producer $p \in P$ is producing a media stream with minimal bandwidth $\mathrm{min_b}(p)$ and maximal bandwidth $\mathrm{max_b}(p)$.

4 Prototype Implementation

A prototype implementation of CoUniverse[1] uses a current stable version of JXTA [7] P2P framework to implement CoUniverse control plane. Both Multiverse and collaborative Universes are implemented as user name and password authenticated private JXTA peer groups separated from public JXTA P2P network. Current implementation of Multiverse lacks most of the functionality mentioned in previous section and is used just for Universe registration and static lookup.

Current prototype implementation of the CoUniverse uses just one AGC to orchestrate all applications within the collaborative universe. We are using a single AGC to simplify the implementation and to avoid synchronization issues between several AGCs running at the same time. We implemented an interface for steering of generic media applications. In the current implementation of CoUniverse, the Universe Peer is able to control a variety of UltraGrid flavors [8] for both uncompressed and compressed full 1080i HD video transmissions—compared to description in [8], bitrates from 250 Mbps to 1.5 Gbps are now also supported, based on several compression and bitrate reduction algorithms. Amongst other supported applications are: VideoLan Client[2] for HDV video transmissions, VIC[3] for low bandwidth videoconferencing (used as a fallback for building of the collaborative environment) and RAT[3] tool for audio transmissions.

[1] Java sources and JAR archive of the CoUniverse are available at https://www.sitola.cz/CoUniverse

[2] http://www.videolan.org/

[3] http://mediatools.cs.ucl.ac.uk/nets/mmedia/

Media streams scheduler was implemented as a constraint-based solver using a Choco solver library[4]. The solver searches for a solution which is a mapping of media streams on particular network links. Formally we are looking for a set of *stream links* $SL = L \times P$. Scheduler plans the stream links so that $(l, p) = 1$ where $(l, p) \in SL$ for a stream link that is planned to be actively used for the data distribution in the Universe and $(l, p) = 0$ where $(l, p) \in SL$ for an unused stream link. For sake of brevity in the text below, we say that stream link (l, p) *exists* iff $(l, p) = 1$.

Speaking in terms of network model given in previous section the constraints for the solver look as follows:

Stream links constraints. Parent network link l of the stream link must have sufficient capacity to transmit the media stream p. Each stream link must have producer or media distributor on its beginning node and each stream link must have a consumer receiving data using the stream link.

Producer constraints. More than one consumer for a particular producer means that there cannot be any direct stream link between consumers and respective producer as the producer has to send the media stream through at least one media distributor.

Consumer constraints. The media stream for each active consumer is received using exactly one stream link. There are no media streams for any of inactive consumers (i.e., those that hasn't requested any data from any producer) and each active consumer has to be covered by the requested producer either directly or through some media distributor.

Data distribution tree constraints. The number of used stream links for producers with only one consumer is greater or equal to the number of producers. That means data may go either directly, or through some forwarding media distributor (typically in case that direct sending is not available for one reason or another). The number of stream links obviously must not exceed the number of all the media distributors in the network plus one. Moreover a minimal number of used stream links is greater or equal to the number of consumers for given producer plus one for a multipoint data distribution.

AE constraints. A single media distributor instance can only serve for distribution of data from a single producer. Any media distributor is not scheduled together with another consumer for the same producer on a single node and there has to be at least the same number of egress media streams as ingress media streams for a particular AE.

Link capacity constraint. A single constraint for link capacities is stressing that the bandwidth requirements of all the scheduled stream links (l, p) must not exceed the capacity of the link l the stream links are bound to.

[4] http://choco-solver.net/

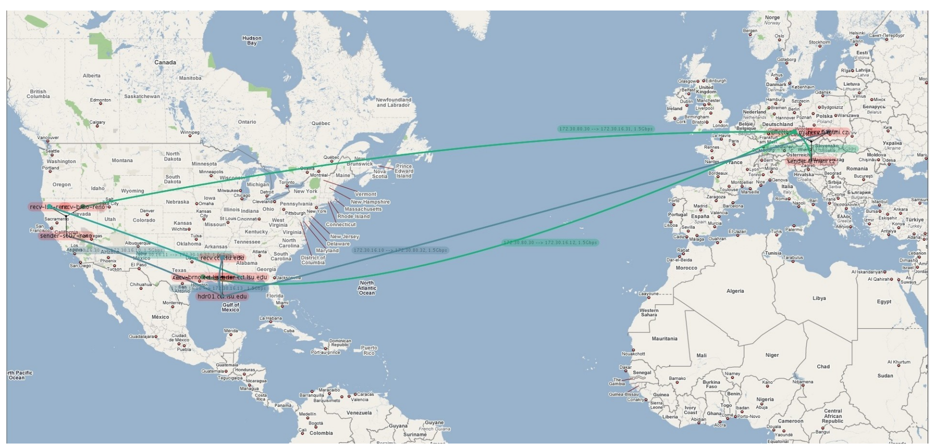


Fig. 2. Visualization of scheduled stream links in CoUniverse during SC'07 demonstration

Another available constraint is based on an elimination of intra-site links (i.e., links (l, p), where node(begin(l)) and node(end(l)) belong to the same site). This can speed up the scheduling up to 10× for many scenarios, but it may also disable some useful solutions, e. g., those where media distributors are collocated in the same site with producers and/or consumers. In case of need, this can be however circumvented by moving media distributors to a separate site.

Based on its settings, the solver can return just a single first match solution or a solution optimized for a minimal media streams distribution latency between the nodes. Based on the network topology and configured media applications the solver may also return a number of equivalent (and even optimal) solutions. In such case the first solution is used and deployed within the collaborative universe.

Because Java lacks any reliable tools for network connectivity monitoring, we have implemented a custom client-server based ping tool. The tool measures not only availability of the peers through the native network, but also network round-trip time, which is an important parameter for latency minimization in our scheduler model. Each universe peer is running the server part implicitly and then is pinging all other known universe peers. In section 3, we mentioned that we need more aggressive monitoring of those network links which are part of some data plane and are used for media applications data exchange than of those network links which are just generally available in the network substrate. This is implemented by a priority and default classes of links which are monitored. A ping client is invoked each second for each network link with scheduled media stream (which is put into the priority class) and each 10 seconds for network links in the default class.

In our prototype, we have implemented a semi-static visualization (see Figure 2) of the collaborative Universe. The visualization is updated with every new scheduling of media streams within the Universe. Currently the visualization

shows only active scheduled media streams with some rudimentary description and static parameters of the media streams. However, even such a simple visualization is helpful to check that the collaborative Universe is started up and configured as was intended to.

5 Prototype Implementation Evaluation

Performance and scalability of the CoUniverse environment heavily relies on the scheduler, therefore we have performed a number of simulations with various network topologies and data distribution schemes and measured the time necessary to obtain a schedule for given network topology and distribution scheme.

We chose a full mesh m:n, 1:n tree and direct 1:1 data distribution schemes as a test cases for evaluation of the scheduler performance and scalability. The network topologies were given by the data distribution schemes and a number of sites in the collaborative universe. The m:n distribution scheme test case topology was generated so that each site had one node with an UltraGrid media application producer a node with UltraGrid consumer for each other site and a node with AE. This scenario simulates full-mesh collaboration among peers. The 1:n tree distribution test case was generated so that one site had an UltraGrid producer node and UltraGrid consumers node for all other sites in the topology, every other site comprised of one UltraGrid producer node and one UltraGrid consumer node. This scenario is realistic, e.g., for virtual classroom type environment, where the lecturer gives his talk in multiple remote rooms in parallel. A corresponding number of AE nodes was generated with respect to the fact that one AE can replicate 1,5 Gbps media stream from an UltraGrid producer to at most 6 UltraGrid consumers where 10 Gbps network link is available. Finally direct 1:1 data distribution was a simple test case with generated pairs of UltraGrid producer nodes and UltraGrid consumer nodes, where each UltraGrid consumer was receiving the media stream from a particular preconfigured UltraGrid producer. This is sort of an artificial scenario to show scalability limits. All nodes had one 1 Gbps and one 10 Gbps network interface configured in all three test cases.

All measurement results were obtained on a 2 GHz Pentium M machine with 1 GB of RAM running a Linux operating system. A 1.2.05 version of Choco solver library was used. Table 1 shows excerpt of measured times necessary to find feasible plans for above mentioned test cases with the Choco solver set up to return all feasible solutions and the corresponding times measured for the Choco solver set up to return just the first feasible solution and exit immediately. The table shows that Choco solver scales reasonably for 1:n and direct 1:1 data distribution schemes with up to 25 nodes in the network topology. The worst scheduler performance was observed for m:n data distribution scheme. For such a scheme we were able to obtain a schedule in a reasonable amount of time for up to 12 nodes aggregated into 3 sites.

Table 1. MatchMaker evaluation

Distribution scheme	Sites	Nodes	Network links	Media applications	Active Elements	Scheduling time (first solution only) [s]	Scheduling time (all feasible solutions) [s]
m:n	2	6	60	6	2	0,308	0,178
m:n	3	12	264	12	3	0,447	0,510
m:n	4	20	760	20	4	1986,047	1970,540
1:n	2	5	40	5	1	0,169	0,181
1:n	4	11	220	11	1	0,285	0,360
1:n	6	17	554	17	1	0,758	0,753
1:n	7	20	760	20	1	0,924	1,110
1:n	8	24	1104	24	2	3,747	8,914
1:n	10	30	1740	30	2	17,518	37,299
1:1	2	4	24	4	0	0,187	0,187
1:1	5	10	180	10	0	0,343	0,333
1:1	8	16	480	16	0	0,862	0,979
1:1	11	22	924	22	0	1,900	2,009
1:1	14	28	1512	28	0	3,382	3,344
1:1	17	34	2244	34	0	5,745	6,160
1:1	20	40	3120	40	0	9,727	10,161

5.1 Demonstrations

A prototype implementation of CoUniverse was evaluated during SuperComputing'07 event and a demonstration at GLIF 2007 meeting. The CoUniverse was used to orchestrate a network of twelve nodes using a high quality, high bandwidth HD video transmissions and audioconferencing to create a multipoint-to-multipoint collaborative environment connecting three sites (Louisiana State University, USA with Charles University, Czech Republic and Academia Sinica, Taiwan).

Creating such a collaborative environment means in praxis configuring and steering of more than two dozens of media applications and Active Elements to bring up the media streams connecting all the sites. Configuring all media applications and AEs at dozen of machines presents a huge amount of manual work which is overwhelming for users of such environment. Moreover there must be at least one user of the collaborative environment having precise idea how to create the media streams between all media applications and AEs based on knowledge of available physical network substrate between all participating sites. Last but not least the users are not able to ensure resiliency and fast recovery of such an environment in case of any network, node or media application failure, because it might mean even newly configuring of all nodes and applications.

Both issues were well addressed deploying CoUniverse. Although creating node configurations for all nodes in the Universe is initially quite time consuming as well, users have to create just a local configurations describing network

interfaces of the local machine and the location of local media applications. The SuperComputing'07 demonstration showed that CoUniverse is also able to respond to changing networking conditions when parts of 10 GbE infrastructure used for the HD video transmissions went down and back up for a couple of times during the demonstration.

6 Related Work

As mentioned in the introduction, some extent of self-organization is usually built into the all but the simplest collaborative tools. H.323 and SIP tools that are considered a sort of industrial standard as a videoconferencing platform can accommodate changes in available link capacity by changing compression parameters of media streams. Isabel [9] platform has similar properties by means of flow server and also features programmable floor control [10], which is however on the level of GUI programmability only.

Probably closest to CoUniverse idea is currently VRVS EVO [11], which allows self-organization of the collaboration network. It is however a closed system that doesn't incorporate external tools and namely it is designed to work only with a low and standard-definition media streams that have bandwidth requirements significantly lower than the link capacity. From user perspective, VRVS EVO can be viewed as a system similar to Skype in terms of both self-organization of the network and usage of low quality media streams.

Another important videoconferencing platform is AccessGrid [12]. AccessGrid is capable of providing high-definition media streams. However, AccessGrid doesn't have any self-organizing properties. The fail-over mechanisms are only very simple and have to be initiated manually by the user, e. g., by selecting unicast media transport instead of multicast. Compared to the other systems, it may seem simple, but it follows several of CoUniverse design principles which the other systems are not compliant with: user-empowered paradigm at least for the collaborative system components (which are open-source and may be installed by end-users arbitrarily) and it is also extensible and incorporates external applications (e. g., UltraGrid to support high-definition media streams).

7 Conclusions and Future Work

In this paper, we have designed a framework for advanced self-organizing collaborative environments called CoUniverse and described its prototype implementation. The system is targeted to incorporating high-end multimedia tools while utilizing advanced high-speed networks with their specialized services.

While the CoUniverse has been designed primarily with the high-end videoconferencing systems in mind, it can be very useful beyond this domain. Any component-based applications with real-time orchestration requirements can be supported. For example, if a scientific instrument, that is generating real-time data, is needed to be incorporated into the Grid infrastructure and the data is supposed to be distributed to one or more locations in real-time, the CoUniverse

can be used to control the data distribution, including the components along the path: data source (i.e., some component that is a direct interface from the instrument to the computer network), data distributors, as well as data receivers (be it storage or real-time visualization systems). It can also allocate dedicated network circuits (e.g., lambda services) prior to starting data distribution and deallocate them after the data transmission is finished. All that is needed to create such an application workflow is to implement a CoUniverse modules for the respective services.

Even though we have already implemented and successfully demonstrated a prototype of the CoUniverse, it still leaves many unanswered questions stated in the introduction to this paper. One big issue is optimization of the scheduling algorithms in order to support larger infrastructures. It should also better utilize knowledge of network structure, even if it is only partial. We want to include scheduling for native multipoint applications. Another issue that needs to be further investigated is programmability of the whole system by its users. This is also important in the context of the scheduler, which may need to be able to incorporate user-defined constraints on its behavior.

Acknowledgments

This project has been kindly supported by the research intent “Parallel and Distributed Systems” (MŠM 0021622419).

References

1. Arrow, H., McGrath, J.E., Berdahl, J.L.: Small Groups as Complex Systems. Sage Publications, Thousand Oaks (2000)
2. Travostino, F., Mambretti, J., Karmous-Edwards, G.: Grid Networks: Enabling Grids with Advanced Communication Technology. John Wiley & Sons, Chichester (2006)
3. MacLaren, J.: Co-allocation of compute and network resources using harc. In: Proceedings of Lighting the Blue Touchpaper for UK e-Science: closing conference of ESLEA Project, vol. PoS(ESLEA)016 (2007)
4. Hladká, E., Holub, P., Denemark, J.: User empowered programmable network support for collaborative environment. In: Freire, M.M., Chemouil, P., Lorenz, P., Gravey, A. (eds.) ECUMN 2004. LNCS, vol. 3262, pp. 367–376. Springer, Heidelberg (2004)
5. Holub, P., Hladká, E., Matyska, L.: Scalability and robustness of virtual multicast for synchronous multimedia distribution. In: Lorenz, P., Dini, P. (eds.) ICN 2005. LNCS, vol. 3421, pp. 876–883. Springer, Heidelberg (2005)
6. Hladká, E., Holub, P., Denemark, J.: An active network architecture: Distributed computer or transport medium. In: 3rd International Conference on Networking (ICN 2004), Gosier, Guadeloupe, March 2004, pp. 338–343 (2004)
7. Traversat, B., Arora, A., Abdelaziz, M., Duigou, M., Haywood, C., Hugly, J.-C., Pouyoul, E., Yeager, B.: Project JXTA 2.0 super-peer virtual network, `http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf`

8. Holub, P., Matyska, L., Liška, M., Hejtmánek, L., Denemark, J., Rebok, T., Hutanu, A., Paruchuri, R., Radil, J., Hladká, E.: High-definition multimedia for multiparty low-latency interactive communication. Future Generation Computer Systems 22(8), 856–861 (2006)
9. De Miguel, T.P., Pavon, S., Salvachua, J., Quemada Vives, J.: ISABEL—experimental distributed cooperative work application over broadband networks. In: Steinmetz, R. (ed.) IWACA 1994. LNCS, vol. 868, pp. 353–362. Springer, Heidelberg (1994)
10. Quemada, J., de Miguel, T., Pavon, S., Huecas, G., Robles, T., Salvachúa, J., Ortiz, D.A.A., Sirvent, V., Escribano, F.: Isabel: An application for real time collaboration with a flexible floor control. In: CollaborateCom 2005 (2005)
11. Galvez, P.: Evo: Enabling virtual organizations. In: CHEP 2007, Victoria, Canada (2007)
12. Childers, L., Disz, T., Hereld, M., Hudson, R., Judson, I., Olson, R., Papka, M.E., Paris, J., Stevens, R.: ActiveSpaces on the Grid: The construction of advanced visualization and interaction environments. In: Engquist, B. (ed.) Proceedings of Simulation and visualization on the grid: Parallelldatorcentrum, Kungl. Tekniska Högskolan, seventh annual conference, Stockholm, Sweden. Lecture Notes in Computational Science and Engineering, vol. 13, pp. 64–80. Springer, New York (2000)

Developing VR Applications for the Grid

Christoph Anthes, Roland Landertshamer, Helmut Bressler, and Jens Volkert

GUP, Institute of Graphics and Parallel Processing
Johannes Kepler University, Altenbergerstraße 69, A-4040 Linz, Austria
canthes@gup.uni-linz.ac.at

Abstract. In the recent years advancements in the development of Networked Virtual Environments (NVEs) can be observed in many domains. Although this technology is available, it is still a challenging task to design and produce these virtual worlds. To ease the creation of such environments the inVRs framework was developed.

Besides this also grid environments have advanced from traditional batch processing systems in the area of scientific computation. Nowadays highly responsive and interactive grid jobs can be supported: Real-time Online Interactive Applications (ROIAs) constitute a potential domain.

The edutain@grid middleware provides an approach to use the advantages of Grid technology, like Virtual Organisations (VOs), dynamic resource allocation, etc. and additionally fulfils the requirements of highly interactive real-time applications.

The rtfOdrom application acts as a prototype application to demonstrate the interconnection between Grid computing and Virtual Reality.

1 Introduction

Multi-user environments with participants all over the world have become more common and are well accepted by the industry. Community platforms like Second Life or multi-player games like World of Warcraft are just a few to mention. Networked Virtual Environments (NVEs) are becoming more and more valuable, for training simulations or collaborative visualisations, since the required graphics and real-time needs can be fulfilled due to the advancement in network infrastructure and the development of graphics boards. Collaborators from all over the world try to solve large scale problem by simulations and parameter studies using distributed computing power.

This distributed computing power has become available through Grid computing [9]. Grid architectures provide many other features than just providing computational resources. They offer for example user and resource management in the form of Virtual Organisations (VOs).

One of the main issues in NVEs and Virtual Reality (VR) applications in general is the real-time interactivity, which is seldomly supported by Grid middleware. Traditional Grid computing architectures provide batch processing

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 352–360, 2009.
© Springer-Verlag Berlin Heidelberg 2009

mechanisms, which can be used comfortably to manage distributed computing power, but they are not usable for interactive data manipulation or visualisation.

The advantages of both domains can be combined by merging functionality of the inVRs framework [2] and the edutain@grid middleware [8]. inVRs is designed to support the creation of efficient NVEs by offering a highly modular architecture and consistent communication mechanisms, while edutain@grid supports Real-time Online Interactive Applications (ROIAs) using a Real-time Framework (RTF) for scalable and interactive communication and the GRIA Grid middleware [15] to support the establishment of VOs.

This paper provides an overview how these different worlds of computational resource management and real-time interactivity can be combined. As an example application rtfOdrom, a VR racing game which is executed on different Grid servers, is described.

The second section gives an overview on the related work, while section three focuses on the architecture of the inVRs framework. The exchangeability of modules and the structure of the network module are drawn out in detail. Section four and five introduce the architectures of the RTF and edutain@grid. The following sections concentrate on the combination of these frameworks and describe rtfOdrom as an example for a real-time application running on the Grid. Finally the last section concludes the paper and gives an outlook into future work.

2 Related Work

Although much research has been done in the fields of Grid computing and Virtual Reality (VR) virtually no approaches exist which try to combine both areas.

Many ways exist for the development of VR applications. Typically three different categories of approaches are common. The first approach is to develop a the VR application from scratch, by using low-level APIs or scene graphs. The second way is the use of fully developed NVEs like DIVE [6] or Graphical User Interfaces (GUIs) like the EON Studio. Finally VEs often make use of frameworks e.g. VRJuggler [12] or DIVERSE [13]. While the first two solutions lack genericity and flexibility, the framework approach can be used to adapt the network capabilities to the needs of Grid computing.

In the area of Grid computing a variety of middleware solutions has been provided. Most of them focus on traditional batch processing approaches. In the CrossGrid project [5] data for a flooding simulation was computed on Grid resources and reduced to real-time display and finally displayed in VR systems like the CAVE [7]. Other approaches like AGJuggler try to make use of the AccessGrid in order to display VEs using Grid resources [11].

The key issues with these approaches is that they either make use of the encrypted communication mechanisms provided by the underlying Grid middleware, or they perform significant offline computation beforehand in order to display a static result or set of results in real-time after the computation. rtfOdrom in combination with the RTF and the inVRs framework allows for non-encrypted communication and thus provides the required real-time capabilities.

3 inVRs Architecture

The inVRs framework offers a clearly structured approach for the design and creation of highly interactive and responsive VR applications in order to improve the development process of VEs and NVEs.

It consists of three independent modules, for interaction, for navigation, and for network communication, two interface layers that allow the abstraction of user input and output display, as well as a system core which stores and manages the state of the VE. An architecture draft of inVRs has been previously described in [2].

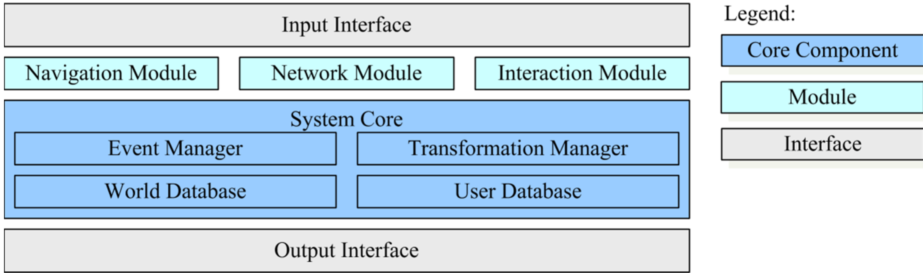


Fig. 1. The Architecture of the inVRs Framework

Figure 1 provides an overview of the inVRs architecture showing the individual components. The flow of data as displayed in this diagram is typically from top to bottom. Input gathered by the devices is parsed by the input interface and exposed to the modules in a data structure describing an abstract controller in order to provide a unified interface. The modules access the abstract controller and generate navigation and data which is processed by the system core managers. The event manger handles discrete reliable events while the transformation manager is responsible for a flow of transformation data packets. More detail on event and transformation management is provided in [1]. Events and transformation data are applied on the user database and the world database which are used to store the state of the VE. The content of these databases is then finally rendered each display frame via the output interface.

In the current implementation inVRs uses OpenSG [14] as a scene graph for rendering the graphical output on a single or multiple stereoscopic displays. Audio output is supported by OpenAL a well known audio library. The input can be retrieved from a variety of sources which could either be regular desktop devices like mice or keyboards, or it can be gathered from VR tracking systems, wands or datagloves.

3.1 Module Exchange

One of the key features behind the design of the framework is its modularity. The need for this becomes clear if we take a look at different application domains.

Some VEs only need a single module e.g. an architecture walkthrough may only need the navigation module. Others such as a networked collaborative safety applications have obviously different requirements. For more interactive VEs like training or phobia treatment applications, both, the interaction as well as the navigation module can be used by interconnecting them through the system core. If an application is designed for multiple users the network module is additionally connected to the core. State changes and entity transformations of the virtual world are in this case transmitted via the network module to the remote participants of the VE.

To achieve this flexibility of module exchange the individual modules are implemented following a plug-in pattern, thus the network module can be easily exchanged in order to support Grid applications. The communication between the managers of the system core and the network module is handled by a high-level interface, which has to be implemented by each inVRs module.

3.2 Network Module

Figure 2 gives an overview on the network module. The communication mechanisms of the system core communicate with the high-level layer of the module via message queues. Geometrical transformations are sent to the network module as unreliable messages and events are sent to be transmitted in a reliable way.

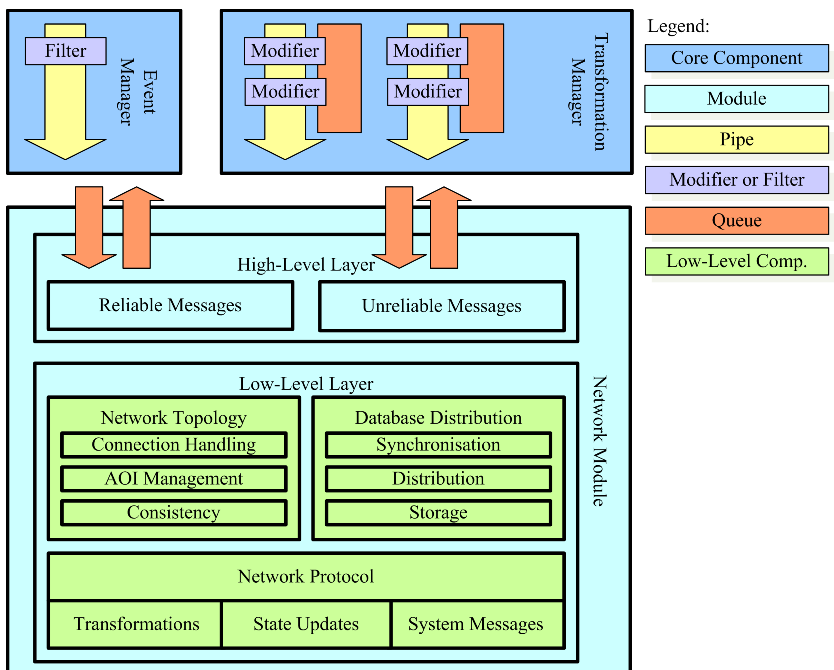


Fig. 2. A High-Level view on the Network Module

The low-level layer shows in a generic view three different aspects which have to be considered for developing an inVRs network application. The message protocol is used for the distribution of geometrical transformations, state updates, and system messages. The network topology handles the connection establishment and communication between the different interconnected nodes, while the database distribution is responsible for storing and managing the states of the virtual world.

This low-level layer of the module has been completely replaced by the communication mechanisms of the Real-time Framework (RTF) in order to be executed on the edutain@grid middleware.

4 RTF Architecture

The RTF was designed to support the scalability of multi-user games by keeping up to the needs of high interactivity.

It is composed of multiple modules which provide controlling, monitoring, communication, streaming, and data storing functionality. Via the controlling module it is possible to startup or shutdown application server instances or migrate server applications from one host to another. The monitoring module can be used to integrate monitoring functionality into application servers. Streaming and data storing functionality are also available for the support of audio and video data transmission and to achieve persistence in networked applications. For the communication between the application clients and the servers the Communication and Computation Parallelisation (CCP) Module is used. This module is integrated into the server and client applications and manages the network connection and communication. It supports different network protocols and provides functionality for the automatic synchronisation of application state information between servers and clients.

The RTF implements a client server architecture which splits up the VE into domains an distributes it over several servers. The VE is populated by entities, which are either static or dynamic objects or avatars representing a single client instance. Entities can travel between domains via portals. On the client side the process of being transferred between domains is transparent to the application even if the destination domain is managed by another server the client's avatar is currently residing on. This is achieved by establishing network connections to potential destination servers when the avatar gets close to a portal to another domain. The migration itself is then achieved by changing the communication destination to the new server.

5 edutain@grid Architecture

The edutain@grid architecture is separated into three different layers, the business layer, the management layer and the real-time layer.

The business layer is the top layer of the system. It is responsible for the management of the different business relationships between the different actors in

the edutain@grid system. From the consumer's point of view this layer provides functionality for login and account management or billing information.

Below the business layer acts the management layer. This layer provides the functionality of resource allocation, monitoring and capacity management. It manages the distribution of the different server applications and supports load balancing mechanisms in order to fulfil Quality of Service (QoS) parameters defined in the business layer.

The third layer in the edutain@grid architecture is the real-time layer. This layer focuses on the real-time communication between the application servers and the client applications. It supports advanced communication functionality like the automatic distribution of game entities or e-learning application data from the servers to the clients. Furthermore this layer supports different network communication protocols and allows to integrate monitoring functionality into the application servers.

The edutain@grid middleware provides interesting features which can be used to support virtual worlds, like a high scalability, which is achieved by the RTF component of the real-time layer. Features like portals provide a user-interface for the end user on many sides of the system (e.g. client, hoster, coordinator).

This middleware has been combined with the inVRs framework in order to connect both worlds.

6 Combining the Three Solutions

In order to interconnect the three approaches the RTF middleware was integrated in the inVRs framework as an individual network module.

The inVRs network module does not follow the principles of a classical client server approach. It is up to the individual modules to maintain a consistent view of the VE of their respective area they are responsible for. The reasoning behind this is that it is more efficient to treat synchronization issues on a per module level rather than on a per application instance level, as different modules may have different needs. The inVRs physics module for example has to distribute the results of the physics simulation to all participants in the VE in order to obtain consistent and vivid object behaviour. However the inVRs architecture does not rule out the possibility of application instances participating in a common network behaving in a different way. In fact the representation of the VE stored locally in the world database of an application instance is designed to be modified by a remote instance without causing any consistency issues. This enables to implement a client server architecture on top of the inVRs framework where the behaviour of entities, both in the RTF and inVRs sense, is organized solely by the server.

Another concept common to the inVRs and the RTF is the idea of having a single user in the VE per application instance. Glinka et al. have described the combination of RTF and legacy applications in [10].

7 The rtfOdrom Example Application

The rtfOdrom is a multi-user VR racing game based on netOdrom, which was originally developed using the inVRs framework, with a simple peer-to-peer interconnection to support two players. A detailed overview of the architecture of the original netOdrom application is described in [3].

In the rtfOdrom is a vehicle corresponds to a RTF-client. The virtual world where the race is taking place is distributed over several RTF servers. The vehicles controlled by the user are represented by users in the inVRs framework. There are also movable obstacles on the racing track which are implemented entities. A physics engine is provided manoeuvring the vehicles and managing vehicle-vehicle and vehicle-obstacle collisions.

The rtfOdrom was primarily designed for testing purposes. Beside focusing on testing RTF components the rtfOdrom was also used to conduct experiments regarding which components of a multi-user application may follow a loose consistency model. Parts of the racing simulation are accessing directly the local world database rather than relying solely on world database updates of the server. In particular it was important to hide the latency associated with user input arising from network lag. Therefore it is inevitable to simulate the vehicle movement within the local application instance which is a behaviour not supported by the RTF framework but corresponds to the approach taken by most inVRs applications.

In order to prevent the clients and servers WorldDatabase state from drifting apart the physics engine has been modified on the client side. Artificial forces and momenta are applied to entities in the local world database in such a way that the state of the server is eventually reached. A description of the implemented physics engine is given by Bressler et. al [4].

8 Conclusions and Future Work

This paper has given an overview of the issues of the combination of Grid computing and VR applications. A brief introduction into the inVRs framework was given which can be incorporated to overcome these issues.

Through the exchange of the inVRs network module with the edutain@grid middleware it was demonstrated that the advantages of grid computing can be used to support interactive and vivid virtual worlds. As an example additional computing resources could be provided in case the computational load on on of the servers used rises above a given threshold.

It is still challenging to display data generated by computational Grid in real-time, but by using automatic geometry reduction mechanisms and storing the post-processed data on ROIA servers it could be even possible to display the data of such applications in real-time.

Acknowledgments

The work described in this paper is supported in part by the European Union through the IST-034601 project "edutain@grid".

References

1. Anthes, C., Landertshamer, R., Bressler, H., Volkert, J.: Managing transformations and events in networked virtual environments. In: Cham, T.-J., Cai, J., Dorai, C., Rajan, D., Chua, T.-S., Chia, L.-T. (eds.) MMM 2007. LNCS, vol. 4352, pp. 722–729. Springer, Heidelberg (2006)
2. Anthes, C., Volkert, J.: Invrs - a framework for building interactive networked virtual reality systems. In: Gerndt, M., Kranzlmüller, D. (eds.) HPCC 2006. LNCS, vol. 4208, pp. 894–904. Springer, Heidelberg (2006)
3. Anthes, C., Wilhelm, A., Landertshamer, R., Bressler, H., Volkert, J.: Net'O'Drom– An Example for the Development of Networked Immersive VR Applications. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4488, pp. 752–759. Springer, Heidelberg (2007)
4. Bressler, H., Landertshamer, R., Anthes, C., Volkert, J.: An efficient physics engine for virtual worlds. In: medi@terra 2006, Athens, Greece, October 2006, pp. 152–158 (2006)
5. Bubak, M., Holger Marten, J.M., Meyer, N., Noga, M., Sloot, P.A.M., Turala, M.: Crossgrid - development of grid environment for interactive applications. In: PIONEER, Poznan, Poland, April 2002, pp. 97–112 (2002)
6. Carlsson, C., Hagsand, O.: Dive - a platform for multiuser virtual environments. Computers and Graphics 17(6), 663–669 (1993)
7. Cruz-Neira, C., Sandin, D.J., Defanti, T.A., Kenyon, R.V., Hart, J.C.: The cave: Audio visual experience automatic virtual environment. Communications of the ACM 35(6), 64–72 (1992)
8. Fahringer, T., Anthes, C., Arragon, A., Lipaj, A., Müller-Iden, J., Rawlings, C.M., Prodan, R., Surridge, M.: The edutain@grid project. In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 182–187. Springer, Heidelberg (2007)
9. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid enabling scalable virtual organizations. International Journal of High Performance Computing Applications 15(3), 200–222 (2001)
10. Glinka, F., Ploss, A., Gorlatch, S., Müller-Iden, J.: High-level development of multiserver online games. International Journal of Computer Games Technology 2008 16 (2008)
11. Gonzalez, D., Arns, L.: Agjuggler. In: I-Light Symposium (September 2005)
12. Just, C.D., Bierbaum, A.D., Baker, A., Cruz-Neira, C.: Vrjuggler: A framework for virtual reality development. In: International Immersive Projection Technology Workshop (IPT 1998), Ames, IA, USA, May 1998. ICEMT (1998)
13. Kelso, J., Arsenault, L.E., Satterfield, S.G., Kriz, R.D.: Diverse: A framework for building extensible and reconfigurable device independent virtual environments. In: IEEE Virtual Reality (VR 2002), Orlando, FL, USA, pp. 183–190. IEEE Computer Society, Los Alamitos (2002)

14. Reiners, D.: OpenSG: A Scene Graph System for Flexible and Efficient Realtime Rendering for Virtual and Augmented Reality Applications. Ph.D thesis, Technische Universität Darmstadt (May 2002)
15. Mike Surridge, S., Taylor, D., De Roure, D., Zaluska, E.: Experiences with gria – industrial applications on a web services grid. In: First International Conference on e-Science and Grid Computing, pp. 98–105. IEEE Computer Society Press, Los Alamitos (2005)

An Information System for Real-Time Online Interactive Applications*

Vlad Nae, Jordan Herbert, Radu Prodan, and Thomas Fahringer

Institute of Computer Science, University of Innsbruck,
Technikerstraße 21a, A-6020 Innsbruck, Austria
{vlad,jordan,radu,tf}@dps.uibk.ac.at

Abstract. The edutain@grid European project [1] is developing a support platform for deployment, management and execution of Real-Time Online Interactive Applications (ROIA) on Grid. In this paper we present an information system designed by the edutain@grid project which provides support for ROIA deployment and monitoring, and offers a generic frontend for ROIA-specific optimisations. We conduct a variety of experiments that justify various decisions of our design, and investigate the performance and scalability of our system with respect to various types of queries.

Keywords: Real-time Online Interactive Applications, Information System, Relational Databases, MySQL.

1 Introduction

The IST-034601 edutain@grid project [1] is focusing on enabling Grid support for general Real-time Online Interactive Applications (ROIA), with particular focus on online games and e-learning applications, including massively multi-user applications embracing large user communities. To achieve this goal, the project classifies ROIA as a new class of Grid applications with the following distinctive features that makes them unique in comparison to traditional parameter study or scientific workflows, highly studied by previous Grid research [2]: (1) the applications often support a very large number of users connecting to a single application instance; (2) the users sharing an application interact as a community, but they have different goals and may compete (or even try to cheat) as well as cooperate with each other; (3) users connect to applications in an ad-hoc manner, at times of their choosing, and often anonymously or with different pseudonyms; (4) the applications mediate and respond to real-time user interactions, and typically involve a very high level of user interactivity; (5) the applications are highly distributed and highly dynamic, able to change control and data flows to cope with changing loads and levels of user interaction; (6) the applications must deliver and maintain certain Quality of Service (QoS) parameters related to the user interactivity even in the presence of faults.

* This research is funded by the IST-034601 edutain@grid project.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 361–370, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Two of the main objectives of the edutain@grid project are automatic deployment of ROIA and load balancing of ROIA sessions by starting new servers or migrating users from overloaded servers to less loaded or newly started ones. To achieve these goals, static information about ROIA deployment procedures and dynamic ROIA session monitoring information needs to be collected and processed. To this end, we designed as part of the edutain@grid management layer an information system where all management services store relevant information about the running ROIA session and the underlying system information.

We present the detailed design of the database schema describing our information system in Section 2. In Section 3 we present experimental results that justify our design and investigate its scalability to various query types. Section 4 concludes the paper and outlines future work.

2 Database Schema

In the following section we present the database schema used by the information system in detail. For performance reasons, we establish no generic schema capable of supporting all types of data structures because such a generic solution would not explore most of the benefits databases provide and would not satisfy type-specific needs. As a result, we define the database schema as a composition of independent, generic, type-specific schemas called from here on *beans*, each bean consisting of one or more customised tables. We describe these schemas in the following sections.

2.1 Host Bean

The host bean is designed to store all ROIA-relevant information about the resources available to the edutain@grid platform (e.g. machines with their connection details). It is defined as a simple tuple of primitive types without any complex nested structures. Since such un-nested types of tuples are exactly the kind of structure a database is working with, it can easily be mapped to a single table. Its schema is shown within Figure 1.

hosts
hostname : varchar(100)
serverStartupPort : int
lowestROIAPort : int
highestROIAPort : int

Fig. 1. Bean table

As stated by the host bean definition, the hostname field is representing the primary key. Since no use cases have been stated by the edutain@grid requirements [3] for querying hosts by anything else than their name, no additional indices have been added.

2.2 ROIA Type Bean

The ROIA type beans are a representation of the ROIA characteristics, completely and uniquely defining individual ROIA such as name, version, interaction complexity, load model, or hardware requirements. Similarly to the host bean, the ROIA type bean can be mapped to the database as shown in Figure 2.

The key is given by a combination of both the name and the version of a ROIA. Since no use case for querying ROIAs by their versions have been stated by the edutain@grid requirements [3], no additional index structure is so far necessary.

roiatypes

name : varchar(100)
version : varchar(100)

Fig. 2. ROIA bean table

2.3 Start-Up Descriptor Bean

The ROIA deployment and start-up information is stored in the start-up descriptor bean which represents a mapping between the resources and the ROIA deployed on them. The start-up descriptor bean has a more complex data structure which, unlike the previous very simple types, is represented through a tuple containing a list of arguments and a map describing the state of environment variables to be set upon execution. Since lists and maps are not supported by databases directly, they had to be decomposed to match the simple tuple-like scheme as requested by any relational database.

When applying standard decomposition rules, any start-up descriptor has to be distributed among three tables. The first table called *startupdescriptor* contains everything defined by the tuple the start-up descriptor is describing without the list and map-like structures, which can be represented through primitive types. The other two tables contain all elements stored within the list and the map, respectively. However, this approach would require a join across all three tables whenever a start-up descriptor has to be read. Further, any resulting set would contain the cross product of the items stored in the list and map structures which potentially produces a lot of unnecessary overhead and increases the result set parsing complexity.

As a consequence, we unified the list and map-like structures into a single table called *startupdescriptorparameter* against common decomposition rules. The downside of this approach might be a slightly bigger disc space consumption caused by potentially unused fields. However, this drawback is rather limited considering the small number of descriptors to be managed. Based on these considerations, the resulting database schema for this data type is as shown in Figure 3.

To be capable of assigning references within the *startupdescriptorparameter* table to the basic *startupdescriptor* table, we add the corresponding primary and foreign keys. As the referential integrity is not checked by the database, any

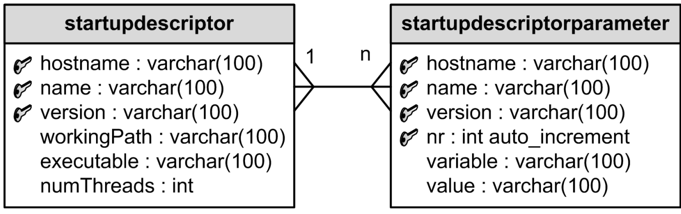


Fig. 3. Start-up descriptor bean table

entry within the argument list will be inserted into the parameter table using the key values of its associated descriptor along with an auto generated index number to ensure a correct reconstruction of the argument order. Environment entries use the name field to represent environment variables and the value field to define their corresponding state.

Based on this schema, a single join is required whenever reading a descriptor from the database. Additionally, the number of rows to be transferred between the database server and client during this read operation is reduced from the product (as it would be based on the original three table approach) to the sum between the number of arguments and the environment variables.

2.4 Record Types Bean

The last type of beans handled by the information system are the record types bean used to store measurement values produced by a service monitoring edutain@grid entities (e.g. ROIA sessions, ROIA servers, resources). The bean records are elements consisting of a single tuple without any nested structures. However, based on the potentially high number of entries and the requirement of providing good performance on insert and query operations, we applied a few special modifications.

A record on its own consists of a metric, a source identifier, a start and end timestamp, a type indicating how the resulting value has been aggregated, and the actual value. We support two record types in a similar way, based on the type of value to be stored. The *simple* record type is supporting a single double value, whereas the *extended* record type supports an array of bytes.

Based on this distinction, two tables each covering all records of a single type would theoretically be sufficient. However, another problem we encountered was to determine the key field ordering. Most queries cover only a single type of metric, which means that the metric should be the first key field and the back-end database tree storing the table content should be sorted according to its value. Unfortunately, this results into out-of-order inserts, since various metrics are getting inserted over time, while experiments showed us that in-order inserts could be executed faster (see Section 3.1). Therefore, we consider that the start timestamp should be selected as the main key element to speed up the insert operations, while the frequent metric-based query is slowed down since the metric-based clustering within the sorting tree is lost.

To overcome this problem and gain advantage of both solutions, we designed a separate table for each metric and, therefore, metric based-clustering can be provided such that entries are sorted according to their timestamps. The main disadvantage is that reading multiple metrics within a single request requires to unify multiple tables. However, since there is no known requirement for such a scenario in the edutain@grid use cases [3], we decided to accept this disadvantage.

Figure 4 shows the resulting table schema where the x symbol within the name of the record table has to be substituted by the unique key of the associated metric (e.g. for the metric name *CONNECTION_COUNT* the resulting

record_metrics
uniqueKey : short
uri : varchar(100)
shortName : varchar(40)
displayName : varchar(100)
valueType : {SINGLE, MULTIPLE}
basicMeasurementUnit : varchar(10)

record_*x*
start : long
end : long
id : long
type : {AVERAGE, CUMULATIVE, INSTANTANEOUS}
value : double / byte[65465]

Fig. 4. Record bean table

record table name is *record_CONNECTION_COUNT*). The *record_metric* table is mainly intended for documentation issues as it is created and updated whenever the system is started but never read. All its information is extracted from an internal, hard-coded enumeration-like class type. The record tables on the right contain the corresponding measurements. Since the starting timestamp has been chosen as the first field within the primary key, quick start time-based range queries are supported and insertions are executed in-order decreasing the insert time.

3 Experiments

As ROIA are very dynamic applications which can generate large amounts of monitoring data in short time intervals, we optimised our information system's performance with a special emphasis on the data storing speed. In this section we report several experiments we carried out to evaluate the performance of our information system implemented on top of the MySQL [4] database platform, which we run on a four dual core processor server with 16 gigabytes of shared memory, a 1000BASE-T network connection, and desktop machines used as clients.

3.1 In-Order and Out-of-Order Insertion

The following experiment evaluates the differences between in-order and out-of-order insertion of monitoring data into our information system. One of the most critical requirements of the information system is the capability to process new monitoring data quickly to fulfill the ROIA real-time QoS requirements. Since monitoring data is usually provided ordered according to some kind of timestamp, the benefits resulting from the in-order insertion should be exploited.

We designed the experiment by generating a random list of five million partially random monitoring entries as described in Table 1. We ordered the resulting list according to the starting time of its entries, memorised it, and used it in a similar way within all successive experiments.

Each experiment starts by creating a new table to store the generated records in the database. For the first run, the table is created using a composed primary key which does not use any of the timestamps as its first component. In our case,

we used the quadruplet *[id, type, start, end]* as key. Afterwards, the generated record list is inserted into the created table in batches of 20 thousand items and the execution time is measured and recorded. After all items have been inserted, the table is cleaned up and the insertion is started again for seven times to eliminate eventual noises.

After the insertion test for the out-of-order key has finished, we continue the experiment by dropping the previous table and recreating it using an in-order key, in this case: *[start, end, id, type]*. The complete test procedure is repeated for the new table and the results are stored. Finally, since MySQL is supporting multiple ways for physical table handling, we covered in this experiment the two most important ones: the index sequential access method (MyISAM) and the InnoDB using a B-tree-based approach.

Table 1. Random data generation

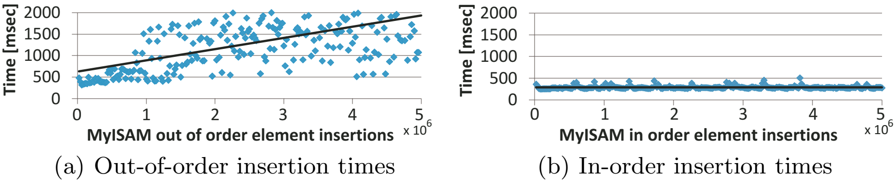
Field	*Value*
Id	Random value $\in [0..99]$
Start time	Linear incremented by 100
End time	Start time + 100
Type	Random value $\in [0..9]$
Value	Random value $\in [0, 1)$

Even though the MyISAM-based databases have a major flow of not supporting real transactions which are required by our information system for data integrity, we still performed this experiment for the sake of performance comparison.

Figure 5(a) shows the results collected using the MyISAM storage engine and inserting elements out-of-order. Every point in the graph represents the average time required to insert the 20 thousand entries in the seven repetitions of the experiment. Obviously, the time required to insert new values is increasing with the number of preexisting elements and becomes quite unpredictable above approximately 750 thousand entries. Therefore, the tables using this storage engine should be limited in size.

Figure 5(b) shows the results of the same experiment with the same storage engine but using a primary key allowing in-order insertion of elements. It can be clearly observed that the time required to insert additional in-order elements is much more stable than for the out-of-order case. The average time of inserting new elements is approximately at the same level as in the best case of the out-of-order insertions. Further, the time required to insert new elements remains constant as the size of the table increases.

The last graph in Figure 6 investigates the impact of the storage engine on this experiment by showing the in-order results of the experiment using the



(a) Out-of-order insertion times

(b) In-order insertion times

Fig. 5. MyISAM insertion times

alternative transaction-safe InnoDB storage engine. The pattern is similar to the MyISAM in-order insertion, although the actual time values are twice as high.

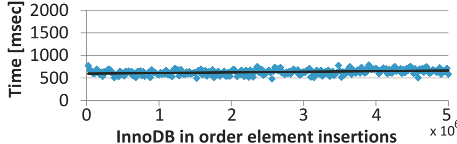


Fig. 6. InnoDB in order insertion times

The corresponding out-of-order experiment using InnoDB produced a pattern similar to the corresponding MyISAM experiment, however, the actual times for inserting new elements were orders of magnitude higher. Because of the slow progress, this experiment was aborted.

3.2 JDBC Usage

The goal of the next experiment is to evaluate the various ways of executing database operations using the Java Database Connectivity (JDBC) toolkit [5]. Most JDBC operations can be performed in multiple ways. For instance, querying information can be performed through ordinary *statement* or *prepared statement* instances, where the latter is potentially caching internally processed compiled versions. While for querying information the decision towards prepared statements is clear (since in this case the query must only be compiled once), for data manipulation operations the problem of choosing the right option remains open.

We designed three types of experiments which we executed for three times using a MySQL server (version 5.0.22) on a remote location through a MySQL Connector/J (version 5.1.6).

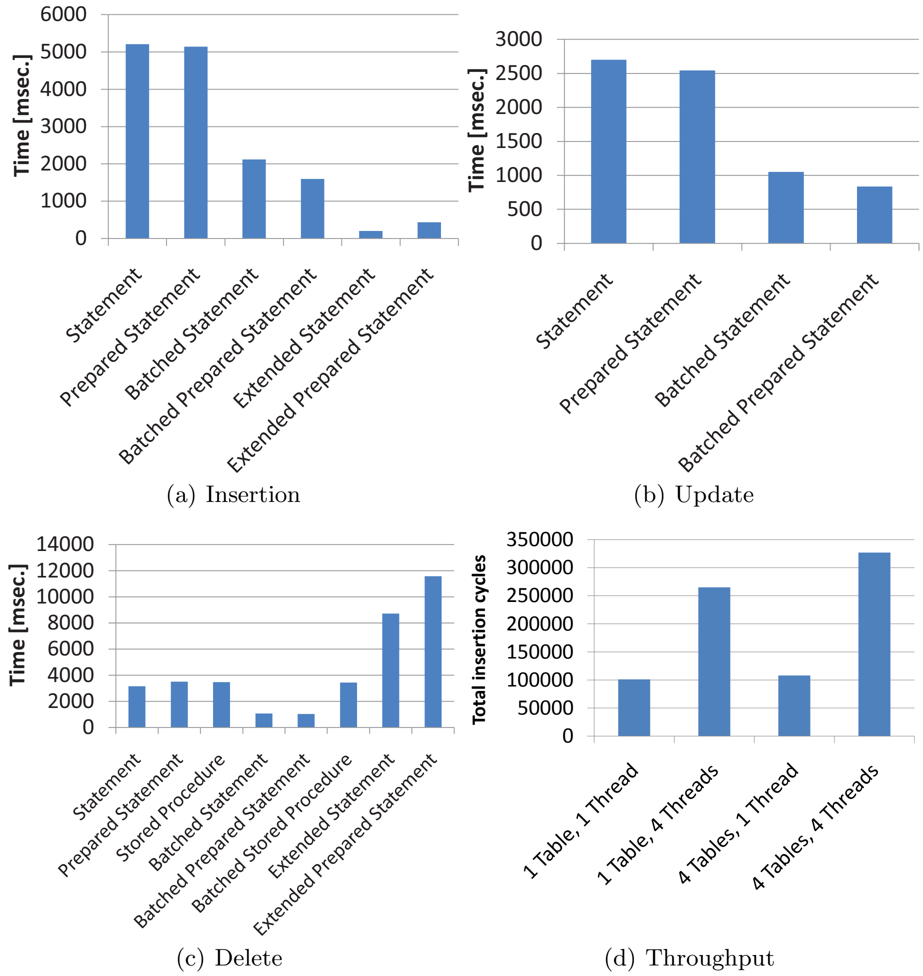
The first experiment concentrates on timing six different techniques of inserting new tuples into a database table. Next to simple statements or prepared statements, we included their batched counterparts, as well as two versions using the extended insert syntax of SQL which inserts multiple tuples using a single call. The experiment starts by creating a new test table containing a key and a value field (both integers), where the key is used as primary key. This step is followed by 10 thousand items inserted using each technique. After each test, the table is cleared to provide equal starting conditions for the next run.

The second experiment performs a similar benchmark for update operations. It first creates and pre-fills the table and afterwards uses multiple techniques to perform 4000 simple update operations on the table to reach a common resulting state. Before each additional technique, the table is restored to its initial state.

Finally, a last experiment performs the same experiment for the delete command. Eight different techniques are deleting 5.000 entries within the same table. The classic statement and prepared statement as well as their batched counterparts are included. Additionally, database entries may be deleted using stored procedures which can be batched too. The pre-compiled operations managed by databases are supported since MySQL version 5 and are intended to reduce the amount of traffic between the database server and client. Further, the extended delete syntax allows to define a *where* clause which indicates the tuples to be deleted and which can be used to specify multiple tuples at once.

Figure 7(a) shows the average times in milliseconds required to insert 10.000 entries. It can be observed that the traditional statements and prepared statements have the worst performance. The sometimes recommended batched version required a reduced execution time and the version using the extend insert syntax turned out to be the fastest. Additionally, although most of the times the prepared version seems to be slightly faster, the simple version is doing better for the extend syntax.

Figure 7(b) shows the average times measured during the update test. Unfortunately, there is no extended syntax for the update statement, however, the batched and the not batched versions of the operations are still supported. Again, the gap between the stand alone and batched variant can be observed, as well as a slight improvement when using prepared statements.



(a) Insertion

(b) Update

(c) Delete

(d) Throughput

Fig. 7. JDBC and parallelism experimental results

Finally, Figure 7(c) shows the results of the delete experiments where an improvement of the batched versions compared to the none-batched once is obvious. However, this effect is not working for stored procedures. The extended syntax does not provide any benefit compared to the other variants.

3.3 Parallelism

The goal of this final rather small experiment was to determine whether there is a difference in accessing multiple tables in parallel using different threads. The reason for this test is the separation of the measurement record bean types among multiple tables, each of them featuring a certain type of measurement. Since most access operations only focus on a single type, the access is reduced to a single table which, combined with the locking mechanism of the database, can speedup access.

For evaluating whether separate tables have an impact on the information system's performance, we developed *load producer clients* which generate high load for specific time intervals (for this experiment we selected a ten minute interval). Each load producer generates ten simple records and five additional extended records with random content, and adds them to the database. This sequence is timed and continuously repeated for the specified (ten minute) time interval. The experiment output is the number of insert cycles completed in the given amount of time. We executed this experiment on one and four tables all scenarios being evaluated using a single, respectively four threads.

Table 2. Speedup and efficiency

	1 thread	*4 Threads*	*Speedup*	*Efficiency*
1 table	103686	264108	2.55	63.8%
4 tables	112604	324685	2.88	72.1%

Figure 7(d) shows the results of this experiment. As expected, the number of completed insertion cycles scales with the number of threads. By increasing the number of tables, the internal locking mechanism is more efficient even for the single threaded version. The total lock cycle count increases by approximately 8%. The results from Table 2 demonstrate that the distribution of the data correlates with an increased efficiency.

4 Conclusions

In this paper we presented the design and evaluation of an information system for ROIA as part of the edutain@grid project [1]. The novelty of our approach is a performance-tunable information system that provides a at the same time a flexible and generic frontend, which makes it suitable for being applied to ROIA. We designed the information system as a relational database on top of the MySQL platform consisting of three main beans: the host bean, the ROIA type bean, and a record types bean. We conducted a thorough set of experiments

for validating our design and for testing the responsiveness and scalability of the system to various kinds of queries.

Our experiments show first the great potential of inserting data in-order into the information system by reducing time required to insert new entries on one hand, and by keeping the data processing time predictable even after insertion of millions of entries, on the other hand. The MyISAM storage engine proved to be faster than the InnoDB in this particular use case, however, InnoDB is the only one providing the required transaction management.

For the insert operation, the JDBC extend syntax provides the best solution although its implementation requires advanced complexity. Since the length of an insert statement is no longer predefined (and therefore limited), it may happen that the overall length exceeds the maximum data package size accepted by the database server. For the update operation, the batched mode of the prepared statements provides the best performance. Fortunately, its realisation does not introduce any additional hazards except the effort of handling transactions. Finally for the delete operations, the result is rather open. The difference between the batched and prepared statements is rather small and may be neglected.

Finally, we observed that the separation of the measurement values among multiple tables does not harm parallel efficiency. However, the internal locking mechanism of the database seems to be capable of handling parallel operations well. Therefore, the data separation on multiple tables appears to have only limited impact. However, the experiment shows that the metric separation does not have any negative side effects when used in parallel too. To investigate the reasons of limiting the parallel efficiency, we need to perform more sophisticated experiments as part of the future work.

References

1. Fahringer, T., Anthes, C., Arragon, A., Lipaj, A., Müller-Iden, J., Rawlings, C., Prodan, R., Surridge, M.: The edutain@grid project. In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 182–187. Springer, Heidelberg (2007)
2. Taylor, I., Deelman, E., Gannon, D., Shields, M. (eds.): Workflows for e-Science: Scientific Workflows for Grids. Springer, Heidelberg (2007)
3. Aragon, A., Fahringer, T., Glinka, F., Lindstone, M., Müller, J., Prodan, R., Surridge, M.: User requirements specification. Deliverable 1.1, IST 034601 edutain@grid Project (March 2007)
4. Atkinson, L.: Core MySQL: The Serious Developer's Guide. Prentice-Hall, Englewood Cliffs (2002)
5. Konchady, M.: An introduction to JDBC. Linux Journal 55, 34–37 (1998)

Securing Real-Time On-Line Interactive Applications in edutain@grid

J. Ferris[1], M. Surridge[1], and F. Glinka[2]

[1] IT Innovation Centre, University of Southampton, UK
{jf,ms}@it-innovation.soton.ac.uk
[2] Institute of Computer Science, University of Münster, Germany
glinkaf@uni-muenster.de

Abstract. This paper presents the analysis, design and implementation of security facilities within the edutain@grid infrastructure, to support secure hosting of Real-Time On-line Interactive Applications (ROIA). The edutain@grid project aims to develop a novel, sophisticated and service-oriented Grid infrastructure which provides a generic, scalable, reliable and secure service infrastructure for ROIA. The class of applications that comprise ROIA have requirements that present obvious challenges to security infrastructure design and implementation. In particular, the requirement to maintain real-time interactivity, particularly within the virtual world subclass of ROIA, precludes a heavyweight solution for securing ROIA. The edutain@grid project is extending 'business Grid' infrastructure that supports Service Level Agreements (SLA) for non-real-time data storage and processing. This infrastructure is based on GRIA and uses Transport Layer Security (TLS) and Web Services Security to secure web service interactions for the provision of data storage and processing. The edutain@grid project is also developing the Real-Time Framework (RTF), which provides communication and parallelisation functionality and API for application developers to create distributed ROIA that can be deployed to and hosted by instances of edutain@grid. The requirements, analysis, design and implementation of security facilities within RTF and the upper business layers of edutain@grid are presented below. We argue that the security facilities provide a suitable compromise between security and performance that will be attractive to the edutain@grid actors and stake holders.

Keywords: Real-time, Grid, Trust and Security, Business.

1 Introduction

Emerging Grid technologies [1] have the capability to substantially enhance on-line games and similar applications. Just as the World Wide Web enables people to share content over standard, open protocols, the Grid enables people and organizations to share applications, data and computing power over the Internet in order to collaborate, tackle large problems and lower the cost of computing.

The edutain@grid project [2, 3] aims to develop a novel, sophisticated and service-oriented Grid infrastructure which provides a generic, scalable, reliable and secure

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 371–381, 2009.
© Springer-Verlag Berlin Heidelberg 2009

service infrastructure for a new class of 'killer' applications of the Grid: Real-Time, On-Line, Interactive Applications (ROIA). ROIA include a broad sub-class of commercially important applications based on virtual environments, including massively multiplayer on-line gaming applications (MMOG), and interactive training and other e-learning applications. The edutain@grid project is aiming to provide an infrastructure to make such applications easier to develop, more economic to deploy and operate, and more capable of meeting the Quality of Experience expected and demanded by end-users.

Grid middleware systems such as Globus [4], gLite [5] and UNICORE [6] enable high-throughput applications by sharing computational resources for processing and data storage to meet the needs of individual and institutional users. ROIA such as multiplayer on-line computer games are soft real-time systems with very high interactivity between users. Large numbers of users may participate in a single ROIA instance, and are typically able to join or leave at any time. Thus ROIA typically have extremely dynamic distributed workloads, making it difficult to host them efficiently. Initiatives such as Butterfly Grid [7] and Bigworld [8] have applied Grid computing to on-line gaming with some success, enabling 'scalable' or 'elastic' terms for hosting such games. However, these 'scalable' hosting services are only as scalable as the hoster supporting them, and typically do not guarantee how far this will be. The edutain@grid project addresses these challenges using 'business Grid' developments such as GRIA [9, 10], but extending them to support scalable, multi-hosted ROIA applications, allowing scaling beyond the limits of any one hoster.

The focus for this paper is the support provided by edutain@grid for secure, real-time communications. In Section 2, we present an overview of the business actors supported by edutain@grid, between which secure, real-time communications must be established. Section 3 describes the edutain@grid architecture and discusses the requirements for secure real-time communication. Section 4 describes how this is addressed using business layer services to exchange keys and set up the required real-time communications. Section 5 describes the security features incorporated into the Real-Time Framework [11]. Finally, Section 6 presents the conclusions of the work described, and discusses possible directions for future work.

2 Business Actors and Value Chains in edutain@grid

To ensure business models for Grid-based ROIA will be economically viable, it was necessary that the edutain@grid infrastructure be generic enough to support a wide range of value chains needed to address different market conditions in these sectors. The analysis revealed an extensive hierarchy of business roles (actors). These include *providers* who host services by which the ROIA is delivered, *consumers* who access the ROIA by connecting to these services and *facilitators* who play other business roles in the creation of ROIA application software, its distribution to providers and consumers, and the operation of ROIA instances. Three important sub-classes of ROIA providers were also identified that must be supported by the project:

- *Hosters*: organisations that host (usually computationally intensive) processes that support a ROIA virtual environment including interactions of users with this environment and with each other.

- *Co-hosters*: other hosters participating in the same ROIA instance; if more than one hoster is involved in a single ROIA, each will regard the others as 'co-hosters'.
- *Coordinator*: an organisation that makes a ROIA accessible to consumers, and coordinates one or more hosters to deliver the required ROIA processes.

Today, on-line game hosters exist, but there are no 'co-hosters' or 'coordinators' because there is only one hoster per game instance. The edutain@grid project removes this limitation, enabling new business models for managing risks of ROIA hosting and delivery, and providing unlimited scalability for ROIA provision. The corresponding value chains are established via bipartite agreements, which also provide the basis for security, following the pattern used in the NextGRID project [12, 13] and with GRIA in the SIMDAT project [14]. The edutain@grid project allows for a wide range of topologies, of which a typical example is shown in Fig. 1:

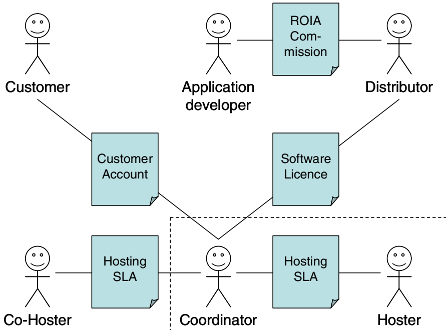


Fig. 1. A typical edutain@grid value chain

In this example, the ROIA application is commissioned by a distributor from an application developer, and operated by a coordinator who also hosts ROIA processes, as shown by the dotted line indicating that the coordinator and one of the hosters are actually the same organisation. Other co-hosters can then be brought in to handle peaks in demand, or if the ROIA becomes so popular that one organisation cannot host it all any more. For a more in-depth analysis of edutain@grid value chains and their implications for SLA terms, see [15]. The challenge for developers of ROIA is to secure real-time communications between the actors in such value chains, bearing in mind that customers will not be certified by a trusted certification authority in advance, and both customers and co-hosters can join or leave the ROIA at any time.

3 Architecture and Security Requirements

The first implementation of the edutain@grid framework was produced in early 2008, and is now being extended to incorporate the security features described in this paper. The prototype focuses on the core edutain@grid actors: the coordinator, the hoster (or co-hoster), and the customer. The framework is based on a Service Oriented Architecture, organised in four layers, as shown in Fig. 2:

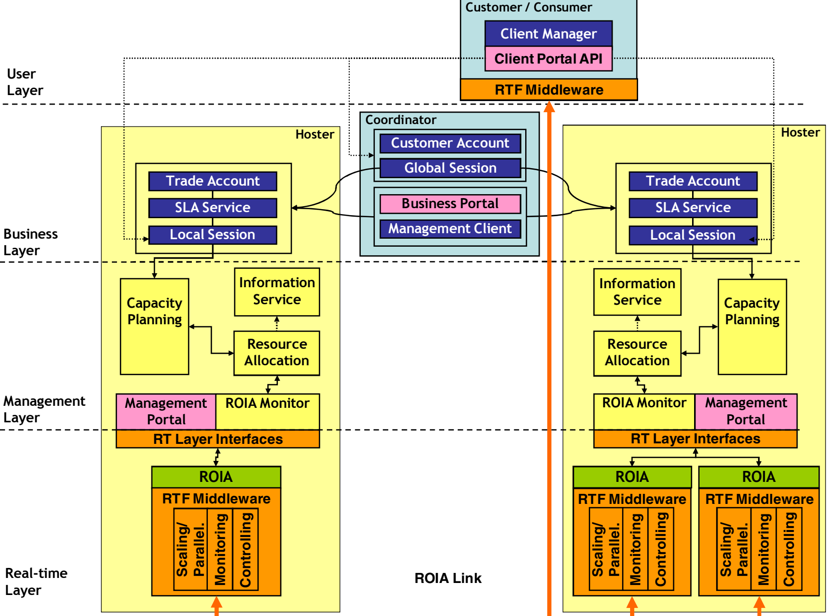


Fig. 2. Prototype edutain@grid architecture

The real-time layer provides a framework [11, 16] for ROIA developers to create scalable applications capable of running across multiple sites. The management layer handles the allocation and management of resources (and ROIA processes) by hosters. The business layer deals with the creation and enforcement of hosting SLAs and customer agreements, including dynamic updating of security policies to ensure ROIA can only be accessed under a valid agreement. There is also a client layer which provides programming interfaces to use services from the other three layers.

The focus for this paper is the establishment and use of secure ROIA links supporting real-time communications; between ROIA clients used by customers and ROIA processes (services) operated by hosters, and also between different ROIA processes even when located at different co-hosters. For more details of other aspects of the edutain@grid architecture and implementation, see [17]. The main security requirements identified are as follows:

- the underlying communication protocol should be based on UDP, since the need for real-time interactivity precludes the use of TCP packet-level handshaking over WAN communication links;
- communications should provide integrity protection at the datagram level, and support authentication of the sender's identity and other attributes, which may be used for authorisation decisions within the ROIA application; and
- communications may provide confidentiality of datagram content: this may not be necessary in on-line games where performance is more important than confidentiality, but is likely to be needed in many e-learning applications.

These requirements have to be met in the context of the business relationships in the ROIA delivery value chain. The business layer of edutain@grid handles this using web services based on management services from the GRIA 5.2 middleware [10] plus some custom ROIA services developed by the edutain@grid project and the real-time layer implementation of secure communications depends on the business layer to handle trust decisions, manage roots of trust and to 'bootstrap' security in RTF.

4 Business Layer Services

The business layer in edutain@grid is responsible for ensuring that only authorised actors can manage a ROIA instance and connect to its ROIA processes. The first step is to establish business agreements and form ROIA value chains between customers and hosters via a coordinator. There are two types of business agreements supported in the current implementation:

- Service Level Agreements (SLA) for hosting ROIA services, established between a hoster and a coordinator;
- Customer Accounts established between a customer and a coordinator to which the customer's ROIA access can be billed.

In each case, the service provider publishes the agreement terms it intends to make available. The service consumer then requests an agreement on those terms, and the service provider can then decide whether to accept or reject the agreement. At present, the decision whether to approve or reject a request is done manually via internal interfaces at the hoster, but in principle, one could automate this by using a business credit checking agency, etc. Normally, the coordinator will form an agreement with at least one hoster and then offer terms to customers, who can open accounts using a similar procedure. The edutain@grid implementation allows customers to open and manage accounts without using a ROIA (e.g. gaming software) client – they could do this using a web browser to access the coordinator's account service, for example.

These agreements provide the roots of trust for securing subsequent actions to launch and access the ROIA itself. Fig. 3 shows the main business layer services and the workflows used to do this. The coordinator first creates a ROIA Global Session resource at its own Global Session Service to hold information (including security policies) associated with each instance of the offered ROIA. The coordinator can then provision the ROIA by creating a ROIA Local Session using a hoster's Local Session Service, which handles the contribution to the ROIA by that hoster. Before creating the requested ROIA Local Session, the hoster checks the credentials of the coordinator with the hoster's own SLA Service. This ensures that only coordinators who have agreed an SLA can use the hoster's ROIA hosting facilities. The coordinator may use several ROIA Local Sessions at different hosters, depending on the scale of resources needed to support the ROIA instance when it starts up. The coordinator then sends them a token validation policy to verify tokens issued for the ROIA Global Session and by each other. All the Local Session Services then signal the edutain@grid management layer to start the ROIA processes, issue them with identification keys, set policies so they can recognise other ROIA processes in the same ROIA (Global Session), and start monitoring the ROIA.

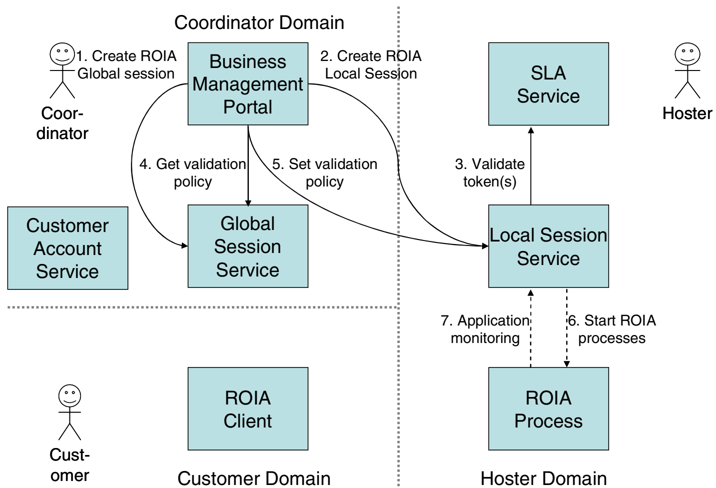


Fig. 3. ROIA Provisioning by the Coordinator and Hoster(s)

At this point, ROIA processes will be running at (potentially multiple) hoster sites, so customers can access the ROIA. The customer signs on to its account via the ROIA client application (Step 1 of Fig 4), obtaining an X509 certificate signed by the coordinator identifying him as a customer. Customers can set up additional pseudonyms at the Customer Account Service, so the certificate need not reveal their true identity, which is helpful in on-line gaming applications where customers rarely use their real names. The ROIA client can then use the certificate to get other short-lived tokens expressing its role in the Global Session (e.g. teacher, student, superhero, etc), and the address of a hoster it should connect to. The ROIA client then contacts the hoster's Local Session Service to get details of the ROIA Process it should connect to. The certificate and other tokens issued by the coordinator match the policies previously sent to the hoster, so the hoster can immediately decide if a connection request should be accepted. These tokens are also short-lived, so it isn't necessary to revoke them. Therefore, the hoster doesn't have to make call-backs to revocation lists, though it does mean the ROIA client has to renew tokens at the coordinator before they expire. By pushing validation policies to hosters in advance, and issuing short-lived tokens that don't need to be revoked, it is possible to use tokens for real-time connections without the usual overheads associated with X509 certification.

The ROIA process then reports the new connection via the hoster's management layer to the Local Session Service. This informs the Global Session Service (if necessary), and reports the usage to the SLA Service, so it can check that the total usage at this hoster remains within the terms of the SLA with the coordinator. At some point, the customer will disconnect, when he stops using the ROIA or when his actions in the ROIA means his connection should move to a different hoster. The Local Session Service is again informed, allowing continued management of the ROIA within the terms of the coordinator's SLA with hosters. It is also possible that the hoster's management layer will at some point predict that the load on its services will exceed the

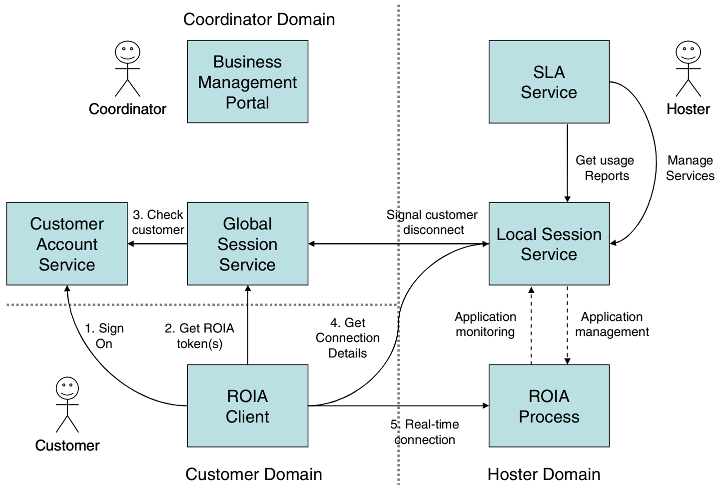


Fig. 4. Accessing The ROIA

SLA limits. That too can be reported via the Local Session Service to the coordinator, allowing 'global' management action to be taken. Actions that could be taken are the transfer of load to another hoster, or to reduce load by disabling access for some customers (if permitted under the Customer Account agreement), before the SLA is breached.

5 Real-Time Framework

The real-time layer in edutain@grid comprises Real-Time Framework (RTF) [11] that supports protocols for using tokens issued by the business layer to control application-level access. RTF is implemented as a C++-library providing parallelization and communication API for application developers to easily develop distributed ROIA.

For securing the communication between the client and ROIA Process, as well as between ROIA Processes, we chose DTLS [18] as it supports encrypted unreliable communication (unlike TLS [19]), can be easily integrated in RTF and needs no operating system support (unlike IPSec). DTLS can also handle the authentication of the communication endpoints during the connection setup.

Fig. 5 shows the main RTF components and the workflows used to establish secured connections and access control using keys and tokens from the business layer:

- the Authentication & Authorization Service (AAS) stores certificates and other security tokens used for the client authentication and authorization;
- the Communication and Distribution Services (CDS) establish RTF connections and manage the parallelization and communication of the distributed ROIA;
- the ROIA is implemented by the application developer on top of RTF CDS; and

- the Policy Decision Point (PDP) intercepts client messages before they are delivered to the ROIA – this is an optional component that can be provided by the application developer or the coordinator.

When the ROIA Process is started, the Local Session Service passes a process-unique identity certificate along with trusted keys and policies for authentication and authorization within the ROIA instance. The ROIA Process stores these for later use in its AAS. A trust chain is thus established from the Coordinator through the ROIA Local Session down to the ROIA Processes.

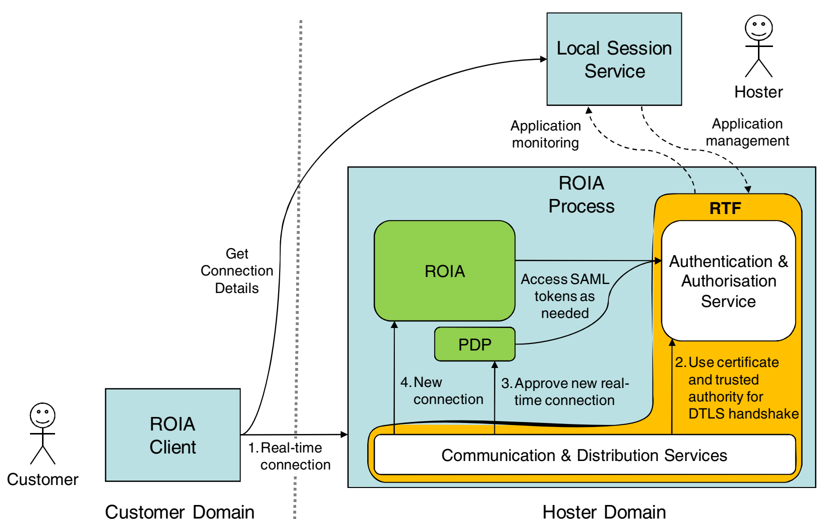


Fig. 5. Creating Real-Time Connections

If a ROIA Client now connects to the ROIA Process (Step 1, Fig. 5) using the connection details that it received from the ROIA Local Session Service, then the RTF's CDS uses the stored certificate and trusted authority for the DTLS handshake and authentication (Step 2). Thus the client and ROIA process can authenticate each other as belonging to the same ROIA instance. Once the handshake is completed, the ROIA Link is associated with the customer's distinguished name (DN) which is based on the login or pseudonym he set up with the Coordinator. If the PDP approves the new connection (Step 3), the ROIA is notified about the newly joined client (Step 4).

Clients communicate with the ROIA Process by sending messages of various, application-dependent types, e.g. to move their avatar, pick up objects, etc. Each message type is identified by a unique integer attribute. An application-specific PDP can be plugged into the RTF during the ROIA Process start-up, and used to determine whether messages should be forwarded to the ROIA. Decisions may be based on the client identity alone, but may also use other user attributes passed with the identity certificate to the ROIA Process in the form of SAML tokens issued by the business layer. These are stored in the AAS for use in subsequent decisions. For example, if a

client can get a SAML token from the business layer saying it is a teacher in an e-learning ROIA, that client would be able to send administrative message types that change the structure of an ongoing e-learning lesson, while other clients could not.

If a ROIA Client wants to gain additional rights within a ROIA, it requests SAML tokens at the Coordinator (Step 1, Fig. 6). These SAML tokens are then pushed by the ROIA Client towards the ROIA Process (Step 2). The authentication and authorization service intercepts these tokens, verifies them against its trusted authority and stores them (Step 3) for later use by the PDP. This way, the client can add rights as needed during run-time by providing additional SAML tokens.

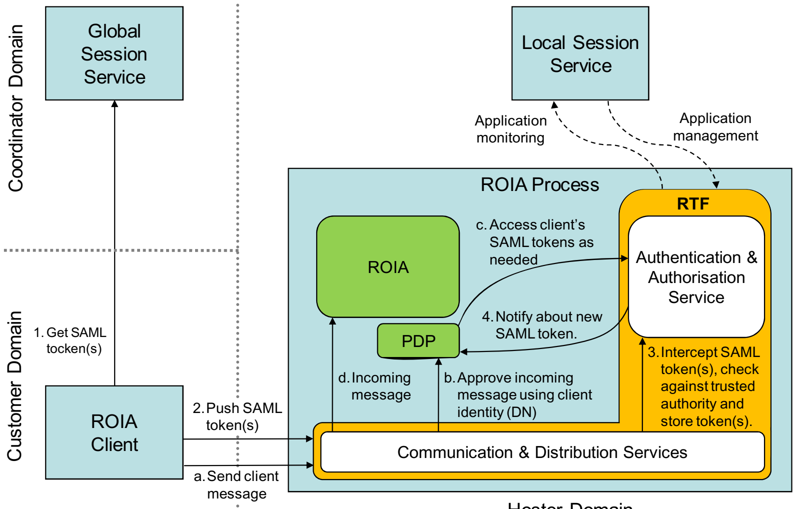


Fig. 6. Application Level Access Control

6 Conclusions and Future Work

The edutain@grid project has developed a framework for grid-based multi-hosting of Real-Time Online Interactive Applications (ROIA), including scalable online gaming and virtual e-learning environments. In achieving this goal, the project has had to address the challenge of securing real-time communications in a way that is consistent with business relationships, by devising a unified architecture for business and real-time communication security. Our approach is based on three main developments:

- the use of Web Services based on GRIA to support the creation of business-level agreements, secured using a dynamic policy implementation in conjunction with Web Service message-level security specifications including WS-Security, WS-Trust, X.509 and SAML [20];
- the use of DTLS for secure real-time data-gram transport within RTF, using trusted keys provided and distributed between actors by the business-level Web Services;

- the use of a simple username-password sign-on procedure allowing users (most of whom are not experts in information security) to obtain and use X509 and SAML credentials without having to manage keys and tokens.

Evaluation studies will now be carried out: testing the performance of secure real-time protocols and the ability of policy/token management protocols to keep up, and evaluating the overall edutain@grid business processes, security, usability, quality of service, etc.

Future work is expected to include the addition of support for configurable RTF security properties (e.g. ciphers, key-lengths, whether to use encryption as well as authentication), so enabling a range of different trade-offs between security and performance requirements. We also plan to investigate options for secure multi-cast RTF communications by extending the key and policy management used to bootstrap RTF security in the business layer.

Acknowledgments. The work described in this paper is supported by the European Union through EC IST Project 034601 'edutain@grid'.

References

1. Foster, I., Kesselman, C. (eds.): The Grid2: Blueprint for a New Computing Infrastructure, 2nd edn. Morgan Kaufmann Publishers Inc., Elsevier, Boston (2004)
2. Fahringer, T., Anthes, C., Arragon, A., Lipaj, A., Müller-Iden, J., Rawlings, C., Prodan, R., Surridge, M.: The Edutain@Grid Project. In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 182–187. Springer, Heidelberg (2007)
3. See the edutain@grid, `http://www.edutaingrid.eu/index.php`
4. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International Journal Supercomputer Applications 11(2), 115–128 (1997)
5. Czajkowski, K., Ferguson, D.F., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W.: The WS-Resource Framework (March 2004)
6. Breuer, D., et al.: Scientific Computing with UNICORE. In: Wolf, D., Münster, G., Kremer, M. (eds.) NIC Symposium 2004. NIC Series, vol. 20, pp. 429–440 (2003)
7. IDC Case Study, Butterfly.net: Powering Next-Generation Gaming with On-Demand Computing, `http://www.ibm.com/grid/pdf/butterfly.pdf`
8. Big World Technology, http://www.bigworldtech.com/index/index_en.php
9. Surridge, M., Taylor, S., De Roure, D., Zaluska, E.: Experiences with GRIA – Industrial Applications on a Web Services Grid. In: e-Science 2005, pp. 98–105. IEEE Press, Los Alamitos (2005)
10. See the GRIA, `http://www.gria.org`
11. Glinka, F., Ploss, A., Gorlatch, S., Müller-Iden, J.: High-Level Development of Multi-Server Online Games. International Journal of Computer Game Technology (August 2008)
12. Snelling, D., Fisher, M., Basermann, A.: NextGRID Vision and Architecture White Paper. Updated periodically: at the time of writing the published version dates from (July 30, 2006)
13. Mitchell, B., Mckee, P.: SLAs A Key Commercial Tool. In: Cunningham, P., Cunningham, M. (eds.) Innovation and the Knowledge Economy: Issues, Applications, Case Studies, IOS Press, Amsterdam (2005)
14. Phillips, S.C.: GRIA SLA Service. In: Phillips, S.C. (ed.) Cracow Grid Workshop, Cracow, Poland, October 15-18 (2006)

15. Ferris, J., Surridge, M., Watkins, E.R.: Business Value Chains in Real-Time Interactive Applications. In: Altmann, J., Neumann, D., Fahringer, T. (eds.) GECON 2008. LNCS, vol. 5206, pp. 1–12. Springer, Heidelberg (2008)
16. Müller, J., Gorlatch, S.: Scaling Online Games on the Grid. In: GDTW 2006, Liverpool, UK, November 15-16 (2006)
17. Ferris, J., et al.: Edutain@Grid: A Business Grid Infrastructure for Real-Time On-line Interactive Applications. In: Altmann, J., Neumann, D., Fahringer, T. (eds.) GECON 2008. LNCS, vol. 5206, pp. 152–162. Springer, Heidelberg (2008)
18. Modadugu, N., Rescorla, E.: The Design and Implementation of Datagram TLS. In: NDSS 2004 (February 2004); See also IETF RFC 4347 on the UDP implementation
19. Transport Layer Security, See: http://www.ietf.org/rfc/rfc4346.txt
20. Web Services Security, See: http://www.oasis-open.org/ committees/ download.php/ 16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

The edutain@grid Portals – Providing User Interfaces for Different Kinds of Actors

Roland Landertshamer[1], Christoph Anthes[1], Jens Volkert[1], Bassem I. Nasser[2], and Mike Surridge[2]

[1] GUP, Institute of Graphics and Parallel Processing
Johannes Kepler University, Altenbergerstraße 69, A-4040 Linz, Austria
rlander@gup.uni-linz.ac.at
[2] IT Innovation Centre, University of Southampton, UK
{bmn,ms}@it-innovation.soton.ac.uk

Abstract. In recent years grid computing has evolved from simple batch processing systems to highly interactive applications. One approach heading in that direction is the edutain@grid project. This project aims to enable the computing power of current grid systems to the area of online computer games and e-learning applications. The shift from traditional scientific computing applications into the domain of business oriented applications has led to the involvement of a variety of actors. All of these organisations and persons have different needs considering the access to the system.

To communicate with the edutain@grid system a novel combination of portals had to be developed. Three types of portals allow the stakeholders to get access to this system. These portals allow a unified and consistent interface approach hiding the complexity of traditional grid applications in order to support users from all different fields to work with edutain@grid in an intuitive way.

1 Introduction

Grid computing [11] has come far in the recent years. Traditional approaches from the area of batch processing have evolved to support interactive applications even in the area of real-time interactivity. An example for the change of Grid applications to support more interactive applications was the CrossGrid project [8], where the computation results of a flooding simulation could be displayed in real-time.

One of the novel approaches to bring more interactivity to the Grid is the edutain@grid project [9] which supports real-time interactive online applications (ROIAs) mostly from the field of e-learning and computer games. The two application domains have commonalities: they support a large amount of concurrent users and they both have to provide instantaneous feedback to the user input. Computational power is needed in these areas mainly for interaction processing of the connected participants of multi-user sessions.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 382–389, 2009.
© Springer-Verlag Berlin Heidelberg 2009

To create support for such applications the portals in edutain@grid follow a different approach than traditional portals. They are considered as a user interface to the different layers and components of edutain@grid, rather than combining and exposing information from different resources in a single view.

Becaues of the real-time constraints of the edutain@grid applications it is not sufficient anymore to restrict portals to web interface technology. Other low-level technology in the form of a C++ API has to be used to interconnect to the interfaces of ROIAs to edutain@grid.

The following Section of the paper will introduce the related work focusing on available portal frameworks. Section three to six describe the different portals for different groups of actors in edutain@grid. The last sections provide an outlook into future work by showing up enhancements of the edutain@grid portals and conclude the paper.

2 Related Work

Portals in the context of Grid computing are typically considered to be web interfaces which aggregate information gathered from different web resources. This composition of information is often enhanced with the possibility of user interaction.

In order to ease the design and implementation of such portals many frameworks have been developed in the recent years. The most prominent examples are Jetspeed [3] and Gridsphere [2]. Other commonly used solutions are jPortlet, uPortal, LifeRay, and PGrade [4] which is built on top of Gridsphere.

A good overview on the mentioned examples can be found in [6], where the previously listed frameworks are described in detail and their functionality and architectures are compared.

The edutain@grid project enhances this portal concept from traditional web portals by providing for example an additional C++ portal API in oder to create a full portal framework which supports the needs of the different actors. A similar approach by offering a portal framework is chosen by Gannon et al. [12].

In the area of computer games Steam [5] could be considered as a community portal where users are able to create accounts and communicate via forums for a large set of games.

3 The edutain@grid Portals

The main approach of the edutain@grid middleware lies in the Grid support for ROIAs. It does not only consider newly developed ROIAs using parts of the middleware but it also includes the support of legacy applications. The main differences of edutain@grid to traditional Grid computing do not only lie in the real-time constraints of the executed jobs, but also in the nature of the life-time of a job.

The portals of the edutain@grid project support the three layers of edutain@grid, namely the business layer, the management layer and the real-time

layer by offering different approaches. They provide a traditional interface approach in the form of web portals for the management layer and the business layer as well as a low-level C++ API based approach as implemented in the client portal API which interconnects with ROIAs executed on the real-time layer.

An early draft on the portal architecture has been previously published by Anthes et al. [7] while more detail on the edutain@grid overall architecture can be found in [9]. This work provides a more precise insight into the developed portals and the interconnections within the edutain@grid architecture.

4 The Business Portal

The business portal is developed for coordinators who manage the distribution of applications on the different hosters. It provides functionality to manage contracts between hosters and the coordinator about provided hosts, SLAs and pricing. The functionality is provided via a standard GRIA web portal which follows the classic web portal approach so that the portal is accessible via an arbitrary web browser. It is directly connected to the business layer of the edutain@grid system which is implemented through GRIA web services [1].

5 The Management Portal

The management portal was developed for hosters which participate in this system by providing computing resources. It is implemented as a web portal which can be accessed via any web browser. The portal provides resource management and monitoring functionality to allow the hosters to keep an overview about their participating machines.

The management portal acts as administration interface for the participating hosters in edutain@grid. It is implemented as web portal using the Gridsphere portal framework. The main functionality of the portal are the configuration of participating machines in the edutain@grid middleware, the display of the status of these machines and controlling functions.

The portal itself is configured to provide three different views to the hosters: the configuration view, the monitoring view and the controlling view.

In the configuration view the hoster is able to manage and configure the machines which should participate in the edutain@grid system. The view consists of three separate portlets, the HostConfigurationPortlet, the ROIATypeConfigurationPortlet and the ROIADescriptorConfigurationPortlet. Each of these portlets allows to modify the configuration for the hoster's local site. The configuration itself is stored and managed by the management layer. The portlets provide a graphical interface in order to modify this configuration via the portal. Figure 1 shows the different portlets of the configuration view.

The first portlet in the configuration view is the HostConfigurationPortlet. This portlet is used to define which machines are available at the hoster's site. It displays a list of all configured machines together with the port settings used for the

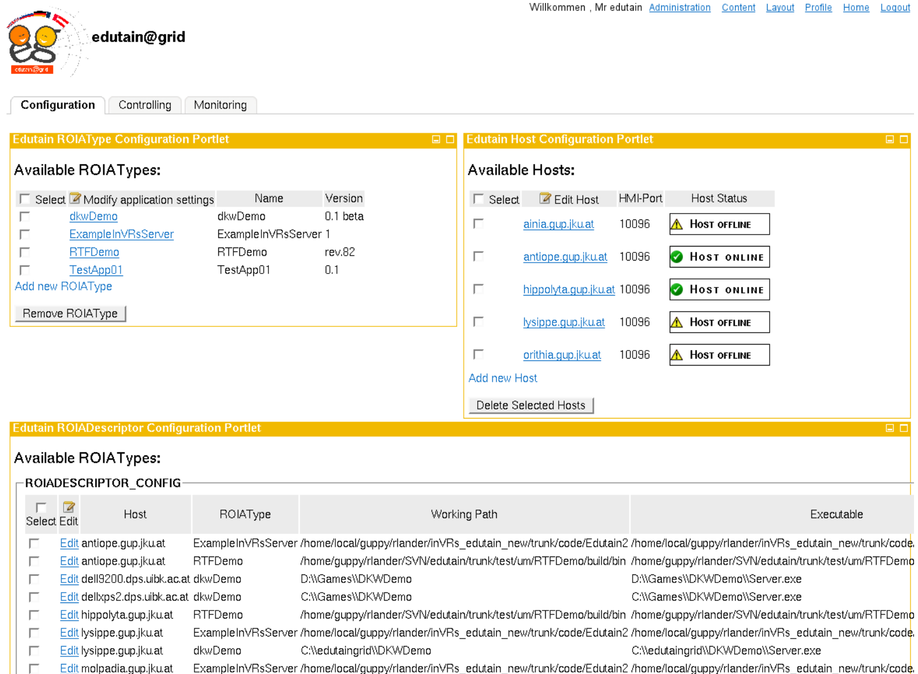


Fig. 1. Management Portal: Configuration view

connection of the edutain@grid management layer to the host. In order to connect the configured machines to the edutain@grid system a separate program has to be started on the single hosts, the ROIAServerStarter. This light-weight application listens on the defined port for commands from the management layer in order to start server applications on the host. To provide an overview if all configured hosts are able to communicate with the edutain@grid middleware the portlet displays a status box for each host which shows if the ROIAServerStarter application is running. Via this portlet it is possible to add new hosts, modify the port settings of participating hosts or remove hosts from the edutain@grid middleware.

The second portlet is the ROIATypeConfigurationPortlet. This portlet allows to define the different application types which can be hosted on the server machines. A configuration for a ROIAType currently consists of the name and the version of the application.

The last configuration portlet is the ROIADescriptorConfigurationPortlet. It allows for the configuration of the deployed applications on the participating hosts. This configuration is used by the management layer in order to start up the applications on the server machines. Every configuration entry currently contains the ROIAType, the host where it is deployed, the working path where the application is installed, the executable which can be started by the ROIAServer-Starter, command line arguments which are passed to the executable at startup time and the number of threads.

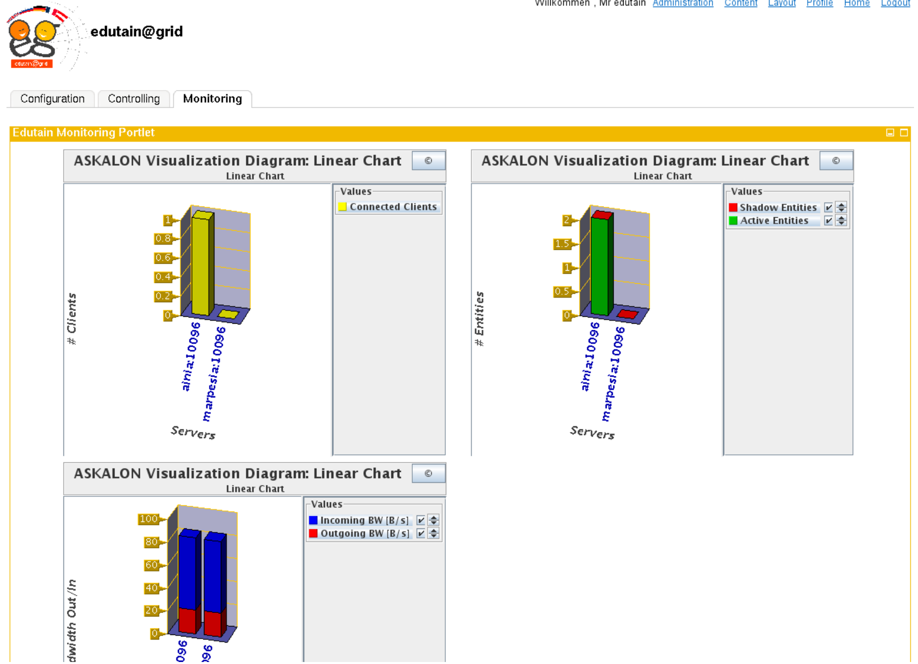


Fig. 2. Management Portal: Monitoring view

The monitoring view of the management portal consist of multiple instances of the MonitoringPortlet. This portlet allows for displaying the visualisation of monitoring data gathered by the edutain@grid middleware. Several monitoring targets like the incoming and outgoing bandwidth, the number of connected users but also application internal data like the saturation of the application loop are provided by the management layer which can be displayed per portlet instance. The configured monitoring data is displayed by a java applet which uses the visualisation library of the ASKALON tool-set for cluster and grid computing [10]. To update the monitoring data in the java applet a monitoring service is running in the management portal. At startup of the java applet a socket connection is established between the applet and the monitoring service. The service streams the monitoring data via this connection to the visualisation applet which displays the data in form of diagrams. Figure 2 shows an example of the monitoring view with three active MonitoringPortlets.

The controlling view of the portal consists of a single portlet, the ControllingPortlet. The portlet allows the user to view which hosts are involved in the different ROIA sessions. Via this portlet the hoster can manually start and stop ROIAProcesses on his local machines. This functionality is intended to be used for testing purposes whenever a new application is deployed on different hosts. The controlling view is shown in Figure 3.

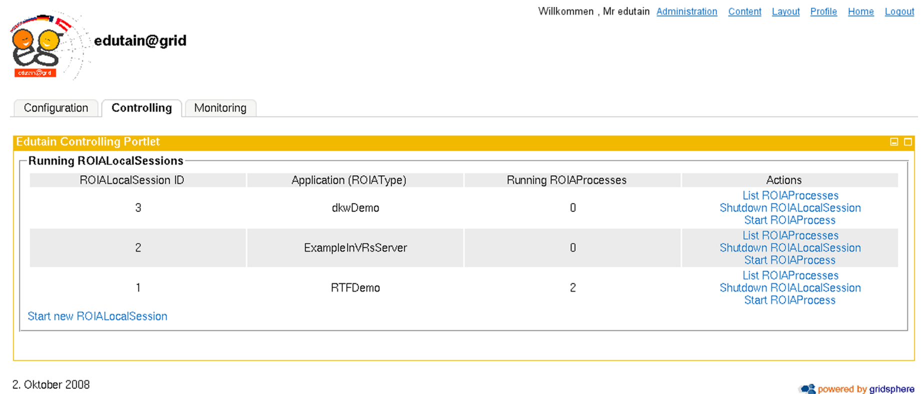


Fig. 3. Management Portal: Controlling view

6 The Client Portal API

The client portal is developed for the end users of the edutain@grid system which are either computer game players or participants in e-learning application. It provides functions to get information about hosted applications, to communicate with other users and to connect to running applications. The portal is implemented as a C++ API which will be included in the application clients to provide a common way of access for the different types of client applications. The communication is achieved via web service calls which allows a secure and standardized way of data transmission.

In order to connect a client application to an application server which is hosted by edutain@grid a communication from the client to the edutain@grid middleware is necessary. This communication is achieved via the client portal. The client portal provides an interface for the client applications to log in to the system, find running application servers, connect to a running server application, etc..

In current computer game clients or e-learning client applications such functionality is provided by an integrated portal. This portal is usually designed to match to the appearance of the application itself. To achieve this property also within edutain@grid clients the client portal functionality is needed to be integrated into these applications. Therefore the portal does not follow the traditional portal definition but is implemented as a C++ library. The API of this library is kept at a high abstraction level so that this portal hides the complexity of the underlying GRIA grid middleware from the application developers and the end users.

To be able to connect to a server application the client portal API provides a function to login to the system. The login function takes a username and password as credential. This information is forwarded via a web service call to the edutain@grid business layer. The business layer then checks the credential for validity and returns a security token if the login was successful. This security

token is later on used in every other communication between the client and the edutain@grid middleware to authenticate the user.

For connecting the client to a running application server a list of available servers has to be provided to the client. Therefore the client portal API provides a method which allows to request all running server instances of a defined ROIAType. The ROIAType is usually preset by the type of the client application in order to find matching server applications. The request is forwarded to the business layer via a web service call. The business layer then checks if the user is allowed to request this information. If so the list of available server applications is replied to the client portal.

After an appropriate server application is selected the client has to get the connection details of the server where the application is hosted. Therefore the client portal API sends an request to the business layer for the IP address and the port of the host for the specified server application. Figure 4 shows the communication chain of this request. At first the client portal API sends a request to the business layer via web service calls containing information about the desired application server. In order to obtain this information the business layer has to communicate with the management layer of the edutain@grid system. This layer is responsible for managing the resources at the different hoster sites. The management layer itself has a connection to all available server hosts. Via this connection the layer can obtain the IP and port information of the server host. This information is returned to the business layer and forwarded to the client portal API as web service call result. The application then gathers the connection details from the client portal API and uses this information to establish the real-time connection to the application server. This example shows that the complexity of the underlying grid communication is successfully hidden from the client application.

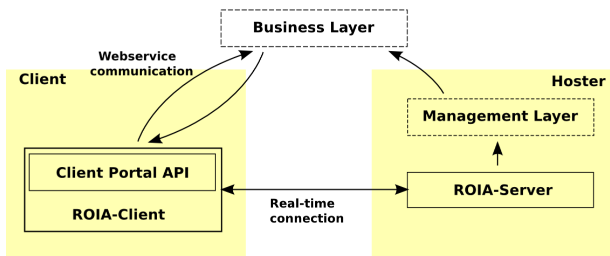


Fig. 4. Client Portal example: Communication path for establishing connection between ROIA-Client and ROIA-Server

7 Conclusions and Future Work

This paper has shown approaches how to support ROIAs in the edutain@grid project by providing a novel portal concept. To offer interfaces for the different

kind of actors the definition of portals has been extended beyond the use of web interfaces.

To communicate efficiently with ROIAs a C++ portal API was provided and described. It is obvious that web portals are not sufficient to support modern computer games, thus Login mechanisms and look-up functionality has been offered in order to be seamlessly integrated into the applications.

To control such applications and to allow for enhanced debugging possibilities on the management layer real-time monitoring, monitoring visualisation and control functionality is offered by the management portal.

In the client portal API community functionality like friend lists or chat should be added in order to support standard game functionality. Packaging the approach of the different portals would lead to a portal framework, which could be used as a generic solution for a class of of interfaces for interactive grid applications.

Acknowledgments

The work described in this paper is supported in part by the European Union through the IST-034601 project "edutain@grid".

References

1. Gria (last visited, October 2008), http://www.gria.org/
2. Gridsphere (last visited, October 2008), http://www.gridsphere.org/
3. Jetspeed-2 (last visited, October 2008), http://portals.apache.org/jetspeed-2/
4. Pgrade (last visited, October 2008), http://www.lpds.sztaki.hu/pgrade/
5. Steam (last visited, October 2008), http://www.steampowered.com/
6. Allan, R., Awre, C., Baker, M., Fish, A.: Portals and portlets 2003. Technical Report UKeS-2004-06, CCLRC e-Science Centre (June 2003)
7. Anthes, C., Landertshamer, R., Hopferwieser, R., Volkert, J.: A novel portal architecture for real-time online interactive applications on the grid. In: 7th Cracow Grid Workshop (CGW 2007), Cracow, Poland, October 2007, pp. 180–187 (2007)
8. Bubak, M., Holger Marten, J.M., Meyer, N., Noga, M., Sloot, P.A.M., Turala, M.: Crossgrid - development of grid environment for interactive applications. In: PIONEER, Poznan, Poland, April 2002, pp. 97–112 (2002)
9. Fahringer, T., Anthes, C., Arragon, A., Lipaj, A., Müller-Iden, J., Rawlings, C.M., Prodan, R., Surridge, M.: The edutain@grid project. In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 182–187. Springer, Heidelberg (2007)
10. Fahringer, T., Jugravu, A., Pllana, S., Prodan, R., Seragiotto Jr., C., Hong-Linh, T.: Askalon: a tool set for cluster and grid computing: Research articles. Concurr. Comput.: Pract. Exper. 17(2-4), 143–169 (2005)
11. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid enabling scalable virtual organizations. International Journal of High Performance Computing Applications 15(3), 200–222 (2001)
12. Gannon, D., Alameda, J., Chipara, O., Christie, M., Dukle, V., Fang, L., Farellee, M., Fox, G., Hampton, S., Kandaswamy, G., Kodeboyina, D., Moad, C., Pierce, M., Plale, B., Rossi, A., Simmhan, Y., Sarangi, A., Slominski, A., Shirasauna, S., Thomas, T.: Building grid portal applications from a web service component architecture. In: IEEE, March 2005, pp. 551–563 (2005)

A Case Study on Using RTF for Developing Multi-player Online Games*

Alexander Ploss, Frank Glinka, and Sergei Gorlatch

University of Münster, Germany
{a.ploss,glinkaf,gorlatch}@uni-muenster.de

Abstract. Real-Time Online Interactive Applications (ROIA) include a broad spectrum of online computer games, as well as challenging distributed e-learning applications, like virtual classrooms and collaborative environments. Development of ROIA poses several complex tasks that currently are addressed at a low level of abstraction. In our previous work, we presented the Real-Time Framework (RTF) - a novel middleware for a high-level development and execution of ROIA in single- and multi-server environments. This paper describes a case study in which a simple but representative online computer game is developed using RTF. We explain how RTF supports the design of data structures and their automatic serialization for network transmission, as well as determining and processing user actions when computing a new game state; the challenge is to provide the state updates to all players in real time at a very high frequency.

1 Introduction

Real-Time Online Interactive Applications (ROIA) form a novel class of technically challenging distributed applications. They include for example e-learning applications, like virtual classrooms, as well as a broad spectrum of online computer games reaching from fast-paced action games to large-scale massively multiplayer online games (MMOG). In order to support high numbers of users, the processing of the application state needs to be implemented in an efficient, scalable manner, e.g., via parallelization and distribution on multiple servers. The communication among participating processes in a ROIA session (clients and servers) needs to be efficient and optimized for highly frequent data transfers. Since generic development approaches able to handle multiple aspects of scalable ROIAs are still lacking, developers implement ROIAs from scratch and at a low level of abstraction, which is error-prone and time-consuming.

The *Real-Time Framework (RTF)* [5] is a novel middleware developed at the University of Münster as part of the European edutain@grid [2] project. RTF simplifies the development process of ROIAs in which users continuously interact and concurrently modify a shared application state. RTF is implemented as a C++ library which is optimized for efficient processing and supports the

* This work is supported by the EU through IST-034601 edutain@grid project.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 390–400, 2009.
© Springer-Verlag Berlin Heidelberg 2009

eventual consistency update model and light-weight UDP communication. RTF enables a high-level development of scalable ROIAs and transparently integrates monitoring and controlling functionality for dynamic resource management. Furthermore, RTF supports various distribution concepts (zoning, instancing and replication) which allow to overcome the saturation of computational and network resources caused by a growing number and/or increasing density of online users.

Our previous work [4] described the high-level concepts of RTF, showed how its distribution mechanisms can be used to implement scalable online games and how the RTF-based approach compares to existing development methods. The process of application development using RTF comprises two groups of tasks: 1) basic tasks like designing data structures to model the application state, distributing the processing between client and server including communication, and introducing new entities, and 2) parallelization tasks of organizing a scalable distributed processing when using multiple servers.

In this paper, we present a case study on using RTF for developing a simple but still quite challenging example ROIA – a multiplayer online computer game. Because of lack of space, we omit the developer tasks needed for the multi-server case; they are left for a future publication. We describe RTF from the developer perspective, in order to show how a particular application can be designed on a high level of abstraction.

The remainder of the paper is as follows. Section 2 describes the fundamental Real-Time-Loop processing model for ROIA and gives a short overview of RTF. Section 3 provides an in-depth view of a development use case for an online computer game. Section 4 describes both the case study and RTF in the context of dynamic resource management. Finally, Section 5 concludes the single-server development case using RTF and outlines the multi-server aspect.

2 Real-Time Loop in Multiplayer Games

The majority of today's online games typically simulate a spatial virtual world which is conceptually separated into a static part and a dynamic part. The static part covers, e. g., environmental properties like the landscape, buildings and other non-changeable objects. Since the static part is pre-known, no information exchange about it is required between servers and players. The dynamic part covers objects like avatars, *non playing characters (NPCs)* controlled by the computer, items that can be collected by players or, generally, objects that can change their state. These objects are called entities and the sum of all entities is the dynamic part of the game world. Both parts, together, build the *game state* which represents the game world at a certain point of time.

For the creation of a continuously progressing game, the game state is repeatedly updated in an endless *real-time loop* [1,9]. Figure 1 shows one iteration of the server real-time loop for multiplayer games based on the client-server architecture. A loop iteration consists of three major steps: At first the clients process the users' input and transmit them to the server (step 1 in the figure).

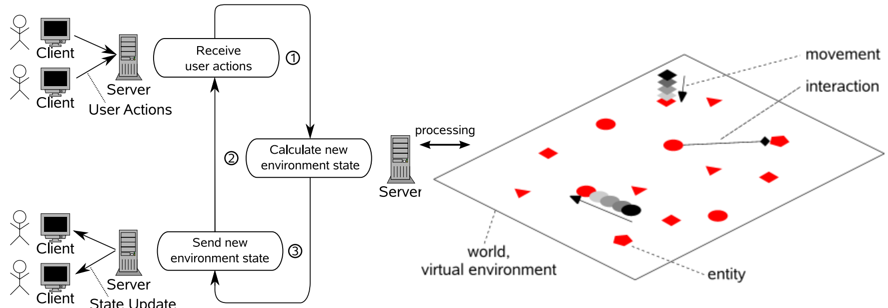


Fig. 1. Real-Time Loop for entity-based ROIAs

The server then calculates a new game state by applying the received user actions and the game logic, including the *artificial intelligence* (AI) of NPCs and the environmental simulation, to the current game state (step 2). As the result of this calculation, the states of several dynamic entities have changed. The final step 3 transfers the new game state back to the clients. The figure shows one server involved in each step, but in a multi-server scenario this may be a group of server processes distributed among several machines.

Figure 2 shows an overview of the use of RTF in a session of a ROIA. The developer implements the application-specific processing following the Real-Time-Loop processing model. This application-specific part can use other application-specific components, like graphics engine, depending on the purpose of the software (client- or server side). To implement the processing, the application developer uses the parallelization and communication functionality provided by RTF. RTF automatically deals with the distribution, both, between client/ server and among multiple servers, and with the communication among all processes participating in a session of the ROIA.

RTF transparently implements the transmission of events between clients and servers and of state updates to clients and other servers (right-hand side of the figure). RTF automatically gains introspection to the application state (performance characteristics) and is thus aware of the current distribution status. This information can be provided to an external, application-independent

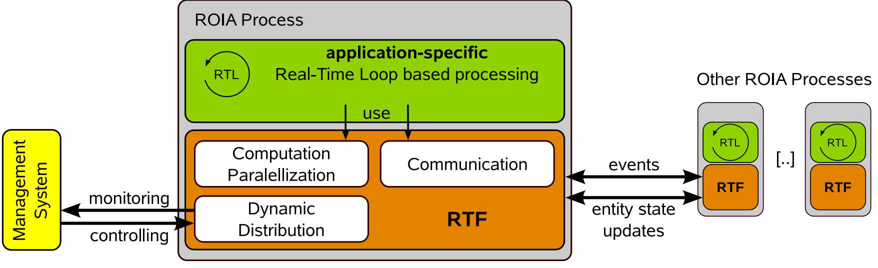


Fig. 2. Overview of the Real-Time Framework used in a ROIA

management system (left-hand side of the figure), which is then able to perform dynamic resource management for the ROIA via RTF. This monitoring and controlling by a resource management system is transparent for the application.

3 Case Study: Development of an Online Game

As a development example we will use a simple online game (*RTFDemo*), which, however, incorporates all fundamental (technical) features of a ROIA:

– Game world is simulated as a 3D virtual environment;
– Each player has one avatar;
– Players can move their avatars (using keyboard and mouse);
– Players interact by shooting other player's avatars (direct hit);
– Entities are solely controlled by the game logic.

To implement the basic ROIA state processing, the developer addresses the following tasks: 1) *data structure design* to model the application state, 2) *application state processing* to distribute the computations between clients and server using events and state updates, and 3) *Area of Interest management*, as well as some general tasks, like creating and introducing new entities. In the following we will describe how these tasks are addressed using RTF.

3.1 Task: Data Structure Design

The dynamic state of a ROIA is usually described as a set of *entities* which represent avatars or non-player characters in the game world. Besides entities, *events* are the other important structure in a virtual environment for representing user inputs and game world actions. Hierarchical data structures for events and entities in complex virtual worlds have to be serializable for efficient network communication.

Describing the Entity State. When using RTF, entities and events are implemented as object-oriented C++ classes. The developer defines the semantics of the data structures according to the game logic. The only semantics of entities that are predetermined by RTF is the information about their position in the virtual world. Entities, therefore, are derived from a particular base class `Local` of RTF that defines the representation of a position for entities. This is necessary since the distribution of the game state processing across multiple servers is based upon the location of an entity in the game world. Besides the requirement of inheriting from `Local`, the design of the data structures is completely customizable to the particular game logic.

In order to enable platform independence, RTF defines primitive data types to be used (e. g., `gcf_int8`). Also, easy-to-use complex data types for vectors and collections are provided to the developer. Overall, more complex entity and event data structures can be easily defined using these primitives.

We start to develop our application RTFDemo from a class to model the state of a player's avatar:

```
 1 class Avatar : public emf::Local {
 2 public:
 3   /* process a new avatar state */
 4   void think(const double& passedSec);
 5   void move(emf::Vector movement);
 6   [..]
 7   DECLARE_SERIALIZABLE_PUBLIC(Avatar, TypeAvatar)
 8 private:
 9   AvatarType avatarType; // type of the Avatar (enum)
10   emf::Vector velocity; // movement
11   emf::Vector orientation; // direction
12   gcf_uint16 health; // cur hitpoints
13   gcf_uint16 maxHealth; // max hitpoints
14   gcf::Annotation annotations; // State changes, Actions
15   DECLARE_SERIALIZABLE_PRIVATE(Avatar) };
```

Listing 1. Class `Avatar` models the state of a player's avatar

Listing 1 shows our class `Avatar` inheriting from `Local` (line 1) in which the position and dimension of the entity are described. Other attributes describe the game-dependent state of the avatar (lines 9-14). Attributes can be primitive types (`health`, `maxHealth`) including enumerations (`AvatarType` in line 9), classes (`velocity` and `dimension`), or even more complex containers of classes (`annotations`). Methods `think` and `move` (lines 5 and 6) implement the modification of the avatar state.

RTF Serialization. RTF provides automatic serialization of the entities and events defined in C++, implements marshalling and unmarshalling of data types and optimizes the bandwidth consumption of the messages. While the developer specifies entities and events as usual C++ classes, RTF provides a generic communication protocol implementation for all data structures following a special class hierarchy. All network-transmittable classes inherit from the base class `Serializable` of RTF. The `Serializable` interface can be a) implemented by the developer, or b) automatically implemented using the serialization mechanism provided by RTF which is generated using convenient pre-processor macros. For all entities and events implemented in this manner, RTF automatically generates network-transmittable representations and uses them at runtime.

Non-entity classes, like actions, are directly derived from `Serializable`, whereas the `Avatar` automatically inherits the `Serializable` interface via `Local`. The `DECLARE_SERIALIZABLE_*` statements (lines 7 and 15 in Listing 1) generate code for the implementation of the `Serializable` interface. `TypeAvatar` (line 7) is a system-wide unique integer to distinguish `Avatar` from other Serializables.

```
1 [..] // application-specific code goes here
2 #include <gcf/GenericSerializerImpl.cpp>
3 IMPLEMENT_SERIALIZABLE_DERIVED(Avatar, emf::Local,
4     ADD_ATTRIBUTE(Avatar, velocity, Unreliable, Public)
5     [..] // dito for all network-transmittable attributes
6     ADD_ATTRIBUTE_DEFAULT(Avatar, annotations) )
```

Listing 2. Implementation of the `avatar`

Listing 2 shows the use of RTF's automatic serialization mechanism. The `IMPLEMENT_SERIALIZABLE` statement (line 3) generates the implementation of the `Serializable` interface. The developer needs to describe attributes that should be transmitted over the network. For example, the `ADD_ATTRIBUTE` statement (line 4) adds the velocity attribute to the description of the serialized form of the `Avatar`. The automatic serialization mechanism can handle delta updates, i.e., only transmitting changed information, and differentiated updates for different processes. In order to use delta updates, the developer tracks modification to attributes in a mask provided by RTF. To use differentiated updates, the developer can specify different types of visibilities for attributes (`Public` in line 4). The developer also specifies for each process its level of visibility.

3.2 Task: Application State Processing

The central aspect of the development approach using RTF is the *real-time loop model* (Figure 1). Most contemporary multiplayer games are based on such a loop whose iterative updates are called *ticks*. RTF allows the game developer to implement his own real-time loop in the well-understood manner and, moreover, provides him a substantial support for implementing and running this loop on both the server- and client side.

The client side needs to 1) determine the user actions, and 2) display the current game state. The server side has to perform 1) processing of the events, and 2) updating of the game state.

Client: Determine User Actions. At first, we read the user's input (from keyboard/mouse). Then we determine the desired action and send it to the server, using the ClientCCPModule (Listing 3, line 5). Serialization and transmission are done by RTF transparently.

```
1   void ClientActionFactory::sendActionMove(){
2   serverPos = mAvatar->getLocation().getPos();
3   emf::Vector newPos = mGraphicManager->getPlayersPosition
        ();
4   ActionMove moveEvent(newPos-serverPos);
5   mClientCCP->sendEvent(moveEvent); }
```

Listing 3. Send a user action via `ClientCCPModule` from client to server.

Server: Process User Actions On the server side, events are automatically received by RTF and appended to the event queue. We process these events and calculate the new game state as shown in Listing 4.

```
1  // emf::EventManager& em = ccpModule.getEventManager();
2  // emf::ClientManager& cm = ccpModule.getClientManager();
3  void  Server::processEvents() {
4  for(emf::Event* e=em.popEvent(); e!=NULL; e=em.popEvent()
       ) {
5    switch(e->getEvent().getType()) {
6    case ActionMove::TYPE: {
7      Avatar &actor = (Avatar&)
8          cm.findClient(e->getSender())->getAvatar());
9      ActionMove& actionMove = (ActionMove&)e->getEvent();
10     actor.move(actionMove.getMovement());
11 } break; } } }
```

Listing 4. Process Events

We access RTF's event queue via the `EventManager` (line 4) and use the RTF type identification to determine the correct class of the event (lines 5 and 6). The move action refers to a specific entity (line 7) which can be retrieved from RTF via the `ClientManager`, which allows to determine the client which has sent an event and get the avatar from that client (line 8). After having determined the event's type and actor, the action is applied to the game state (line 10).

Server: Process New Game State. At this step, the active entities are updated accordingly to the game rules and game logic. As our implementation applies move actions directly to the entity state, we do not have to move players' avatars in this step anymore. But we have to update the rest of the game state, e. g., to move the non-player characters:

```
1  void Server::updateAllEntities() {
2    std::map<gcf::DGObjectID, emf::Local*>::const_iterator
         it
3      = om.getActiveObjects().begin();
4    for(; it != om.getActiveObjects().end(); it++) {
5    switch(it->second->getType()) {
6      case Avatar::TYPE:
7        Avatar& avatar = (Avatar&) *it->second;
8        avatar.think(ticklength); // let every active
9        [..] } } } // process other types of entities etc.
```

Listing 5. Update all Entities

Server Real-Time Loop. The complete processing cycle for the server is shown in Listing 6. During the `onBeforeTick` (Listing 6, line 2) and `onFinishedTick` (line 6) calls, RTF fills the event queue and sends the state updates to the clients. Therefore, the game application should not modify the game state concurrently to these calls. The call in line 6 processes the AoI Management which we will deal with in section 3.3.

```
1  while(!serverQuit) {
2    ccpModule.onBeforeTick(); // inform RTF about begin of
        tick
3      processEvents();
4      updateAllEntities();
5      interestManagement.update();
6    ccpModule.onFinishedTick(); // inform RTF about end of
        tick
7    [..] } // sleep, check for server quit etc.
```

Listing 6. Server Real-Time Loop

Client Real-Time Loop. The real-time loop on the client side looks similar to the one on the server side, but works with a specific client-side version:

```
1  while(mInputProcessor->mContinue) {
2    // sleep, calculate time since last tick
3    mClientCCP.onBeforeTick();
4      // Capture input and send actions (e.g.,
          sendActionMove)
5      mInputProcessor->capture(inputTimer.getTicklength());
6      updateEntities(timeSinceLastTick); // apply state
          updates
7      mGraphicManager->renderFrame(); // render a frame
8    mClientCCP.onFinishedTick(); }
```

Listing 7. Client Real-Time Loop

After determining user actions and sending them to the server (line 5), newly arrived updates from the server are processed (line 6) and the new game state is displayed on the screen (line 7). The client loop is completed by surrounding `onFinishedTick` (line 8) and `onBeforeTick` (line 3) calls, during which incoming events sent by the server are enqueued and game state updates are applied.

3.3 Task: AoI Management

An *Area of Interest (AoI)* concept assigns each avatar in the game world a specific area where dynamic game information is relevant and thus has to be transmitted to the avatar's client. AoI optimizes network bandwidth by omitting irrelevant information in the communication. RTF supports the custom implementation of arbitrary AoI concepts by offering a generic publish/subscribe interface. The engine continuously determines which entity is relevant for a client and notifies RTF of each change of an "interested" relation through a `client.subscribe(entity)` and `client.unsubscribe(entity)` call. RTF automatically takes care that the entity is available/ updated at the client or removed from it. RTF automatically replicates a new entity to other processes (clients or servers) according to the AoI management. Clients will be informed about (dis-) appearing entities via the `ClientCCPModuleListener` interface.

```
1 void Client::objectAppeared(emf::Local& obj) {
2 switch(obj.getType()) {
3   case gcf::TypeAvatar: {
4     Avatar& avatar = static_cast<Avatar&>(obj);
5     mGraphicManager->AvatarAppeared(avatar); break; } } }
```

Listing 8. Notification about new entities (client-side)

Listing 8 shows the implementation of the `objectAppeared` callback for RTFDemo. During this callback the application can perform procedures to handle the newly appeared object, e. g., preparing the entity for introduction to the graphics engine (line 5).

3.4 General Tasks: Client Connection and Entity Creation

A general task that occurs independently of the continuously state update is introducing new entities to the application state when they are created. A typical example for introducing new entities is a client connecting to the session or a newly spawned NPC. RTF informs the application about connecting clients with the `clientConnected` callback of the `ClientListener` interface.

```
1 // Place a walking NPC in the world.
2 Avatar& npc = *new Avatar(Avatar::ZONE_TRAVELER,
3     emf::Space(2400, FLAT_HEIGHT, 750, 40.0f, 85.0f, 40.0
        f),
4     emf::Vector(50,0,0), emf::Vector(1,0,0));
5 ccpModule.getObjectManager().registerActive(npc);
```

Listing 9. Introducing new entities to the application state

Listing 9 shows how new entities can be introduced to the application state. The server creates a new instance of the `Avatar` class (line 2) and registers this new entity with RTF by invoking the `registerActive` method of the `ObjectManager` (line 5).

4 Benefits of Using RTF

After all the described basic tasks are solved and the desired game logic is implemented, our RTFDemo game appplication is ready to operate online sessions with multiple users using a single server. Fortunately, RTF's development and runtime support for ROIA goes beyond this single-server case. RTF supports multi-server distribution of the application state processing to implement scalable ROIAs and, furthermore, dynamic monitoring and controling of ROIAs during runtime. This allows to operate ROIAs using distributed resources and, with the possibility of allocating additional resources on peak-loads and increasing resource usage efficiency, as described in [6].

RTF supports the ROIA distribution approaches *zoning*, *instancing*, and *replication* on a high level. Each of these approaches allows to scale a different aspect of a ROIA. for example, zoning scales the overall size of the application state,

i.e., the number of users, by identifying independent parts of application state. First scalability experiments that demonstrate the performance of RTF's zoning support are covered in [6]. Another work [7] evaluates the scalability of the First Person Shooter game Quake 3 using RTF's replication support and compares the performance of the original Quake 3 with the version using RTF.

5 Conclusion and Related Work

The main novel features of our RTF middleware are as follows: (1) Highly optimized and dynamic real-time communication links adapt to changes in the dynamic distributed environment and can automatically and transparently redirect the communication to new servers; (2) Hidden background mechanisms allow the runtime transfer and redistribution of parts of a game onto additional resources without noticeable interruptions for the users; (3) A high-level interface for the game developer abstracts the game processing from the location of the participating resources; (4) Monitoring data are gathered in the background and used by a management system for capacity planning.

Our case study has demonstrated how an example online game can be developed using RTF on a high level of abstraction. Some game development studios re-use existing solutions, e. g., successful game engines like *Unreal* or *Quake*, or use optimized libraries for particular tasks like network communication. When using only a communication library, like *Torque Network-Library* [3], or *HawkNL* [8], developers have to build data structures and serialization mechanisms from scratch, while using an existing engine requires the use of predefined entities and events, which reduces flexibility. In contrast, RTF provides an optimized high-level entity and event concept enabling automatic serialization while still providing full design flexibility. In a future publication, we will cover in detail how RTF solves additional implementation tasks for multi-server processing, like distribution and scalable parallel processing, transparently for the user.

References

1. Dalmau, D.S.-C.: Core Techniques and Algorithms in Game Programming. New Riders Games (2003)
2. Fahringer, T., Anthes, C., Arragon, A., et al.: The edutain@grid Project. In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 182–187. Springer, Heidelberg (2007)
3. GarageGames. Torque network library, `http://www.opentnl.org/`
4. Glinka, F., Ploss, A., Gorlatch, S., Müller-Iden, J.: High-level Development of Multiserver Online Games. International Journal of Computer Games Technology Article ID 327387 (2008)
5. Glinka, F., Ploss, A., Müller-Iden, J., Gorlatch, S.: RTF: A Real-Time Framework for Developing Scalable Multiplayer Online Games. In: NetGames 2007, Melbourne, Australia, September 2007, pp. 81–86 (2007)
6. Gorlatch, S., Glinka, F., Ploss, A., et al.: Enhancing Grids for Massively Multiplayer Online Games. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 466–477. Springer, Heidelberg (2008)

7. Ploss, A., Wichmann, S., Glinka, F., Gorlatch, S.: From a Single- to Multi-Server Online Game: A Quake 3 Case Study using RTF. In: ACE 2008, Yokohama, Japan (December 2008) (to appear)
8. H. Software. HawkNL, `http://www.hawksoft.com/hawknl/`
9. Valente, L., Conci, A., Feij, B.: Real Time Game Loop Models for Single-Player Computer Games. In: SBGames 2005 (2005)

Abstractions for Distributed Systems (DPA 2008)

The computing infrastructure of tomorrow will be very different from that of yesterday. In this rapidly evolving landscape, the development of applications that can remain neutral to underlying infrastrutural changes remains a challenge. The goal of the Workshop on Abstractions for Distributed Systems is to try to address how utilizing distributed systems can be made easier via the use of abstractions — support for commonly occurring patterns, which could be either programming patterns, application usage patterns or infrastructure usage patterns.

This workshop aimed to determine where programming abstractions are important and where non-programmatic abstractions are likely to make greater impact in enabling applications to effectively utilize distributed infrastructure. The workshop had a balance of applications and topical infrastructure developments (such as abstractions for Clouds).

Shantenu Jha
Dan Katz
Manish Parashar
Omer Rana
Murray Cole

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, p. 401, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Co-design of Distributed Systems Using Skeleton and Autonomic Management Abstractions*

M. Aldinucci[1], M. Danelutto[1], and P. Kilpatrick[2]

[1] Dept. Computer Science, Univ. of Pisa, Italy
[2] Dept. Computer Science, Queen's Univ. of Belfast, UK

Abstract. We discuss how common problems arising with multi/many-core distributed architectures can be effectively handled through co-design of parallel/distributed programming abstractions and of autonomic management of non-functional concerns. In particular, we demonstrate how restricted parallel/distributed patterns (or skeletons) may be efficiently managed by rule-based autonomic managers. We discuss the basic principles underlying pattern+manager co-design, current implementations inspired by this approach and some results achieved with a proof-of-concept prototype.

Keywords: Algorithmic skeletons, design patterns, distributed programming abstractions, autonomic computing, grids, clouds, multi/many core.

1 Introduction

The development of parallel and distributed programs is recognized to be a challenging task. The management of the concurrent activities and communications together with the non-functional concerns, such as performance, security, fault tolerance, all require substantial efforts during both the design and implementation phases and in debugging, tuning and maintenance of the application.

The sustained evolution in parallel and distributed architectures makes application development even harder, as technological improvements and architectural model changes must be catered for. On the one hand the increasing prevalence of multi- and many-core systems necessitates the use of some kind of parameterisation of the code to support hundreds or even thousands of parallel activities, as it is inconceivable that a programmer may design, implement and manage hundreds or thousands of different activities. On the other hand, the emergence of first grid and now cloud architectures, with their inherent heterogeneity and dynamicity, has thrown into stark relief the burden of handling non-functional concerns.

Researchers in two distinct areas have tried separately to tackle these issues, but to date there is not a comprehensive methodology to attack the distributed

* This work has been partially supported by EU FP6 NoE CoreGRID, EU FP6 STREP GridCOMP and Italian FIRB Insyeme projects.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 403–414, 2009.
© Springer-Verlag Berlin Heidelberg 2009

application problem in general. On the one hand, *algorithmic skeletons* provide programmers with higher-level abstractions that can be used as building blocks for complex parallel and distributed applications [6]. This addresses the need for structuring. On the other hand, *autonomic computing* has, with some success, provided means to manage some non-functional aspects important in parallel and distributed applications, such as those related to performance (self-configuration and self-optimization) and fault tolerance (self-healing) [13].

In this paper we present an approach which is based on combining algorithmic skeletons and autonomic computing. It advocates a structure/management co-design approach to system development. Skeleton and autonomic management abstractions are given, and the required interfaces allowing interaction between the two are discussed. Refinements to component-based and services-based implementations are then described briefly and results presented.

The proposed approach also provides an attractive separation of concerns between system and application programmers. System programmers have responsibility for providing suitable skeleton frameworks taking care of issues such as process communication, etc., *and* also for ensuring appropriate management of non-functional concerns. This frees the application programmer to focus on selecting a suitably parameterized skeleton and supplying the core functional code; and, for specifying non-functional concern requirements via, for example, some sort of service level agreement (SLA) (although this latter remains a considerable challenge and is currently only achievable to a modest degree).

The rest of the paper is as follows: Section 2 introduces abstractions for the basic skeleton and autonomic computing concepts, Section 3 proposes a methodology for the co-design of computation structure and autonomic management, Section 4 describes an implementation derived using this methodology and presents experimental results achieved within the GridCOMP project. Section 5 explores the challenge of multi-concern management and Section 6 concludes the paper.

2 Programming Abstractions

We introduce here two "generic" programming abstractions: one to capture the structure of a parallel/distributed application and one to deal with non-functional concern management. The co-design of these two abstraction will eventually lead to a much more powerful and effective programming abstraction, whose preliminary implementation and results are briefly outlined in Sec. 4.

2.1 Structuring Abstractions

Successful parallel and distributed applications usually implement some well-known and efficient parallel or distributed computation *design pattern* [16]. These patterns can be recognized as useful distributed programming abstractions to be implemented and optimized once and for all, and then provided to the application programmers, in such a way that the effort of developing efficient distributed applications is factorized across several similar application designs. A natural choice to provide such abstractions to the application programmer

is the *algorithmic skeleton* concept [7,6]. An algorithmic skeleton is a parametric, reusable, efficient implementation of a commonly used parallel/distributed computation pattern. The application programmer can pick up an algorithmic skeleton, instantiate it with suitable code and data parameters and obtain immediately a working application. Depending on the skeleton framework available, algorithmic skeletons can be (more or less) arbitrarily nested to obtain increasingly complex parallel and distributed applications.

Consider a classical skeleton, often used to model distributed computations running on classical distributed architectures, such as COW/NOWs and grids: the divide and conquer pattern. This pattern can be abstracted as a higher order function:

$$(D\&C\ t\ b\ d\ c)\ x = \texttt{if}(t(x))\ \texttt{then}\ b(x)\ \texttt{else}\ c((\texttt{map}(D\&C\ t\ b\ d\ c)(d(x)))$$

where the parameters represent: the function deciding if a termination case has been reached ($t : \alpha \rightarrow \texttt{boolean}$[1]), the function computing the base case ($b : \alpha \rightarrow \beta$), the function splitting a non-base case into sub-cases ($d : \alpha \rightarrow [\alpha]$) and the function combining the results of sub-cases ($c : [\beta] \rightarrow \beta$). By providing the appropriate parameters the user can obtain the working divide&conquer function $(D\&C\ t\ b\ d\ c) : \alpha \rightarrow \beta$. For example, in order to get a working D&C sort function the user should provide a t indicating when the sorting has to be performed sequentially, a b sequentially computing the sort for base cases, a d for splitting (long) lists into a list of (shorter) sublists and finally a c function for combining ordered sublists into an ordered list. All the details relating to the actual computation of the sort according to the divide&conquer pattern are hidden (or "embedded") within the $D\&C$ higher order function.

More generally, complex and richer sets of skeletons are provided that allow the user to express a computation as a skeleton/pattern composition. The following core (abstract) skeleton set has been defined (in slightly different forms) in a large number of skeleton frameworks, including P3L [4], Muesli [14], Lithium/muskel [3,8], SkeTo [15], ASSIST [2] and Calcium [5]:

$$S = seq(C) \mid farm(S) \mid pipe(S,S) \mid map(S) \mid reduce(S)$$

$$C = \langle some\ function\ code\ in\ any\ suitable\ host\ language \rangle$$

In this case, $farm$ denotes the embarrassingly parallel, stream apply-to-all pattern, *pipe* denotes the usual stream parallel computation in stages, *map* and *reduce* model the corresponding data parallel collective operations, and, finally, *seq* just wraps sequential code in such a way it can be used as a parameter in another skeleton.

These higher-order functions can be provided in a way suitable for to the programming model adopted by the user. For example, as library objects for OO programmers, or as composite components or plain services for component and service-oriented programmers.

[1] We denote with $f : \alpha \rightarrow \beta$ the type of a function processing items of type α to produce results of type β.

Algorithmic skeletons may be used to raise the level of abstraction presented to the programmer of parallel and distributed applications by abstracting (and confining in the implementation level) all those aspects not directly related to the function the programmer wants to compute and to the *kind* of parallel/distributed patterns to be exploited. Those details are dealt with in the implementation of the algorithmic skeleton and thus do not directly concern the application programmer. Furthermore, some quantitative aspects that have an impact on the skeleton implementation and, as a consequence, on its performance, can be abstracted through parameters, thus allowing easy skeleton tuning by the application programmer or, as alternative, viable ways to control skeleton behaviour in the implementation (i.e. in the compiling tools and/or in the run time support). For example, consider the parallelism degree. This could either be one of the parameters provided by the application programmer as a kind of SLA when instantiating the skeleton, or it could be a parameter completely managed by the implementation. In the former case, the programmer may make several test runs before identifying the "optimal" parallelism degree for his application, possibly without the need of recompiling the application[2]. In the latter case, the run time system may adjust the parallelism degree upon recognition that the performance of the application does not fit that predicted by the abstract performance model.

2.2 Management Abstractions

When managing parallel and distributed applications, several non-functional concerns such as performance, fault tolerance, security and adaptivity may require consideration on an on-going basis with little or no input from the user. These concerns can be handled at two different levels: either directly at the application code level or within some autonomic manager interacting with the application code. In the former case, the burden lies completely with the application programmer; in the latter case, it becomes a system programmer concern. Furthermore, in the latter case many more autonomic aspects can be included in the manager, making it potentially even more effective, leveraging on the fact that it is implemented as an independent activity. In both cases, however, what typically has to be implemented is a control loop:

$$(\texttt{AM}\ m\ a\ p\ e)(C) = (\texttt{AM}\ m\ a\ p\ e)(e\ (p\ (a\ (m\ C)))\ C)$$

where m represents the monitoring actuated on the current computation, a represents the analyse activity identifying a suitable policy to be executed, p is the function providing plans to implement a given policy and finally, e is the execute function, applying plans to computations (C) in order to adapt the computation according to the chosen policy.

As in the case of structuring abstractions, if users are provided suitable abstractions modelling this kind of autonomic controller, they can obtain running,

[2] Assuming the compiled application will run with some kind of `-np`, MPI-like parameter.

optimized autonomic manager by specializing the general, second order, recursive function with appropriate parameters.

For example, consider performance tuning in an embarrassingly data parallel computation. In this case, the user may provide a monitor function computing current throughput (time spent computing a single data item and time spent to retrieve input data and to deliver (partial) results), an analyse phase that will consider whether the grain limit for this computation has been reached (i.e. the grain such that the time spent to deliver input data to and retrieve results from remote computing elements equals the time spent to compute the data item locally), a plan phase determining either to increase or to decrease the allocated computational resources and finally an execute function applying the planned activities on the current computation to implement policy decisions.

Autonomic managers may be used to raise the level of abstraction presented to the programmer of parallel and distributed applications by abstracting all those aspects directly related to management of their non-functional concerns. To enhance further the abstraction level presented to the autonomic manager designers/implementors, we found it beneficial to express the autonomic cycle behaviour through *business rule*s rather than via the functions mentioned above. In this case, the system programmer is given a set of monitoring and actuation actions, that can be used to get measures about the current computation and to implement adaptive actions, respectively. Then he may completely customize the autonomic manager behaviour by providing *if-then* rules where the *if* part is a first order predicate on the monitored values and the *then* part corresponds to the plan/execute component of the control loop. The autonomic manager periodically scans the (prioritized) rules available, identifies the *fireable* ones (those whose *if* predicate evaluates to *true*) and finally applies the adaptive actions specified in the corresponding *then* part.

3 Structuring and Management Co-design

In this section we propose a co-design approach to developing the structure and management of distributed systems. The aim is to devise a methodology for identifying and implementing distributed programming abstractions which model parallel/distributed computation patterns and handle non-functional features. This methodology may be used to make available programming abstractions that i) can be used (i.e. instantiated according to the general programming model chosen[3]) to implement applications matching exactly the particular parallel/distributed pattern defined by the abstraction, and ii) take care of relevant non-functional aspects via autonomic managers, where the non-functional requirements are specified by the user via a SLA.

In order to be able effectively to co-design distributed application structuring and non-functional concern management, we must identify first suitable interaction patterns between the two. In other words, we have to establish which plays an active role and which a passive role, and how the active actor may impact upon the

[3] OO, component based, service based, ...

passive counterpart. It seems natural to consider the managers as being the active entities and the skeletons the passive ones as autonomic managers are devoted to taking decisions that have to be applied in the execution of an application.

The second step concerns identification of the kind of malleability supported by the passive actor. This requires definition of those parameters which may be controlled from outside an algorithmic skeleton, and thus, as a consequence, which adaptive actions can be ordered by the manager. Triggering of the adaptive actions requires definition of the observable measures of the skeleton that can be sampled via monitoring and, in turn, definition of the policies and plans to be considered in the manager. If the skeleton does not provide suitable mechanisms to monitor relevant parameters then, no matter how good the abstract performance model we have in the manager, it is not possible to perceive in the manager that the computation is not performing as expected and thus trigger some adaptation action. Similarly, if the skeleton implementation does not provide effective actuation mechanisms, then the manager cannot implement any kind of corrective policy. Therefore the skeleton implementation *must* provide appropriate monitoring and actuation interfaces for the manager.

Skeleton malleability can be achieved via appropriate parameters. Thus we will assume that each of the skeletons considered has more that just the function parameters outlined in Sec. 2.1. In particular, we will consider that each of the skeletons has a parameter specifying the *parallelism degree* of the skeleton itself. In addition, a skeleton may have a boolean parameter stating whether the communications involving that skeleton should be secured or not. Whether this should be an actual parameter or some kind of meta-data (e.g. provided via annotations) is beyond the scope of this work. With these parameters available, one can easily envisage managers interacting with running skeletons by setting/resetting those parameters via appropriate setter/getter methods provided by the skeleton interface.

The factors listed determine the nature of the co-design that can be used for structuring skeletons and autonomic managers. It is clear that the more effects we want to control via the manager, the more monitoring and actuators methods should be implemented in the managed skeletons. It is equally clear that the better interaction we have among manager and managed entities, the better autonomic management policies we can implement.

The methodology mentioned at the beginning of this section, can thus be summarized as follows.

1. First, the skeleton set used to structure our application is identified.
2. Then the malleable skeleton interface is designed and implemented allowing i) monitoring of the measures of interest for implementing the autonomic management policies and ii) actuation of the decisions taken by the autonomic manager.
3. Finally, autonomic manager control loop is implemented that i) gathers the relevant monitoring values from the malleable skeleton, ii) activates the rule engine and iii) finally executes the fireable rules through the actuation mechanism interface of the malleable skeleton.

The result will be a programming framework where application programmers build applications by instantiating the skeleton/manager programming abstractions with suitable parameters and devote to the implementation[4] most (all) of the cumbersome activities needed to develop efficient, autonomically managed parallel/distributed applications.

4 Implementation and Results

The approach described in Section 3 has been experimented with in several projects, and the table in Fig. 1 recalls the distinguishing features of the corresponding prototypes. The more important experiments have been made in the framework of the CoreGRID FP6 NoE while designing the GCM (Grid Component Model) and then within GridCOMP, the spin-off FP6 STREP aimed at providing an open source reference implementation of GCM.

	User programming model	Skeletons supported	Manager implemen-tation	Handled non functonal concerns	Policies	User contracts	Skeleton monitoring/ actuator interface
GCM BS	*Component based*	*Farm. Pipe, Dataparallel*	*Inner component*	*Performance*	*Java code then drools*	*Java code then drools*	*AC controller*
SCA service	*Service based*	*Farm, Pipe (partial)*	*Inner service*	*Performance*	*Drools*	*Drools*	*Bean*
muskel	*OO (library)*	*Farm, Pipe (user expandable)*	*Object with callbacks*	*Fault tolerance*	*Java code*	*Java object*	*RMI + local objects*

Fig. 1. Different features of several co-designed skeleton+autonomic manager frameworks (GCM Behavioural skeleton framework [1], SCA service autonomic task farm experiment [9], muskel full Java skeleton library [8])

In this context, the *behavioural skeleton* concept has been developed [1] within the reference GCM implementation built on top of ProActive middleware [17]. A behavioural skeleton (BS) is a component modeling a common parallelism exploitation pattern on parallel and distributed architectures and providing an autonomic manager taking care of the performance non-functional aspects related to the parallelism exploitation pattern considered. In GCM, task farm and data parallel behavioural skeletons have already been implemented and the implementation of a pipeline behavioural skeleton is undergoing. The task farm BS models embarrassingly parallel computations, the data parallel BS models several kinds of data parallel computations, including those sharing a state among their parallel activities, and the pipeline BS models computation in stages. All the current behavioural skeletons handle only performance issues in their autonomic managers.

[4] To the system programmers, but this activity is needed just once, when the skeleton/manager pairs are designed and implemented.

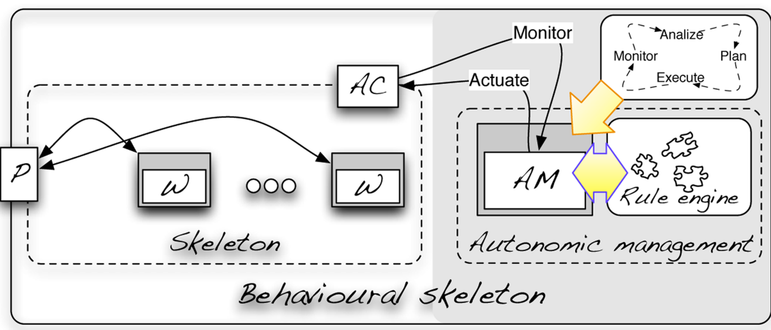


Fig. 2. Abstract schema of Behavioural skeleton: P represents the functional interface, the grey part represents component "membrane", i.e. the non-functional part of the composite GCM component, AC is the the autonomic controller providing a monitoring and actuator interface to the manager. The W are the inner components whose parallel/distributed interaction is managed by the skeleton.

Autonomic managers in behavioural skeletons are implemented using a JBoss rule engine [12][5]. Rules establish manager policies. The precondition part uses methods provided by an *autonomic controller* (AC) bean associated with the implemented skeleton (see Fig. 2). Actuation mechanisms are provided also as methods of the same AC bean, and they are called while executing the action part of fireable rules. The rules are evaluated in a control loop: once the execution of the currently (higher priority) fireable rule action part is terminated, the evaluation of the precondition part starts again. Rules currently included in the autonomic managers allow increase and decrease of the resources allocated to a BS in such a way that a user supplied *performance contract* (SLA) is ensured in the presence of variations in the load and availability of the computing and internetworking resources used to run the application. Performance contracts, in turn, are expressed in terms of throughput via JBoss rules submitted (statically, at the beginning of the application execution, or dynamically, while the execution progresses) to the autonomic manager of the BS.

Separation of concerns is achieved as proposed in Section 3 as behavioural skeletons are implemented by system programmers and application programmers need only choose one of the available BS and provide the appropriate parameters to get a fully working, performance optimized application.

Using the GCM BS prototype we developed several synthetic applications and GridCOMP partners developed more realistic use cases, including biometric identification and fluid-dynamic parameter sweeping applications [11]. Typical results achieved with the GCM BS are shown in Fig. 3. The plot relates to an

[5] Jboss uses *Drools*, that is "a business rule management system (BRMS) with a forward chaining inference based rules engine, more correctly known as a production rule system, using an enhanced implementation of the Rete algorithm" according to Wikipedia (`http://en.wikipedia.org/wiki/Drools`)

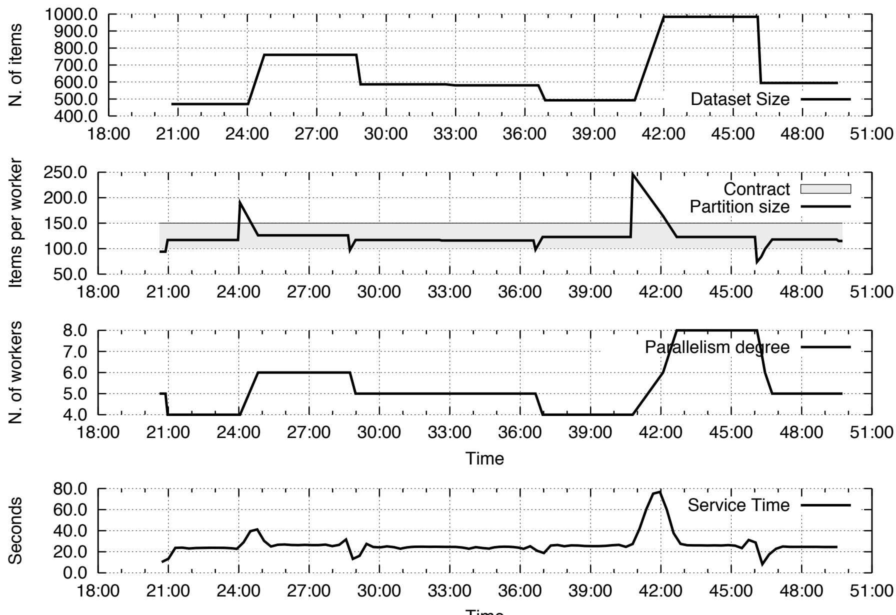


Fig. 3. GCM behavioural skeletons at work

application looking up items in a databases. Several user provided databases are supplied, each with its own stream of items to be searched. Comparing input items with an item in the database takes a non-negligible time. The user-specified contract requires that each worker should have between 100 and 150 database entries to compare, in order to get a reasonable response time (grey bar in the second graph). The dimensions of the databases supplied are plotted on the first graph and the actual database partition size in workers is the line plot of the second graph. The autonomic manager of the data parallel BS reacts by adding and removing workers (third graph) in such a way that the requested partition size varies within the contract range and an acceptable service time is achieved (fourth plot). These results have been achieved by running the application on three different architectures: a Fast Ethernet NOW, GRID 5000 [10] and an SMP multicore core architecture (up to 8 cores). In all cases, the BS autonomic managers reacted as expected and performance has been adapted to the varying load conditions of the target architecture. Experiments are ongoing that demonstrate that the same results can be achieved when heterogeneous NOWs of single and multi-core machines are targeted.

These results assess the concepts and methodologies discussed here. Furthermore, in [9] we discussed a similar implementation providing a *WorkPool* service computing independent tasks according to a task farm skeleton, whose execution is managed by a *WorkpoolManager* service using a JBoss drools engine and interfacing (monitor and actuator interfaces) the managed skeleton to the manager

via a suitable interface bean. With this prototype[6] we demonstrated that the co-design methodology described in Sec. 3 can be easily exported to other programming frameworks (the service framework in this case) while preserving the programmer's investment and fulfilling the "minimal disruption" requirement stated by Cole in his skeleton "manifesto" [6].

5 Multiple Non-functional Concern Management

While the proposed approach has been shown to be effective for a range of underlying programming paradigms, there remain significant challenges, not least in addressing systems where multiple non-functional concerns are to be managed simultaneously. For example, the manager may be required to handle performance, security and fault tolerance aspects of the pattern/skeleton at hand. Thus, in the general case, monitoring may involve different, possibly independent values, independent policies may exist relative to the different non-functional concerns and, finally, decisions taken in relation to different policies may be somehow inconsistent or even conflicting.

Therefore some *meta* policy may be needed to handle autonomic management of different non-functional concerns. The simplest such policy is the weighted one. Different non-functional concerns are given a weight (or a priority) and either those with higher weight/priority are considered first (i.e. the corresponding policies are considered and the corresponding actions taken) or, in the case of policies whose effects can be somehow "scaled" a weighted policy effect is considered (i.e. policy i actions are executed with weight w_i). However this strategy cannot be applied in the general case, as in the general case it makes no sense to execute an action "with weight w_i". We need more complex strategies, and these strategies can probably best be implemented using some business rule engine such that used to implement the rules relating to autonomic management of a single concern.

In this case, we have to distinguish rules used to implement *intra*-non-functional concern policies (*ground* rules) from those implementing *inter*-concern policies (*meta* rules). In addition to normal priority-based handling of rules, system programmers should be able to exploit meta-rules *before* actually actuating fireable ground-rules, but *after* knowing which exact ground rules are fireable and the relative priorities.

For example, consider the case where both performance and security are being managed. Suppose we have a rule stating that we can add more resources to the current computation if it is under performing, and another stating that a resource can be managed without the need to use secure communications and that both are fireable. Application of either rule will probably increase the performance of the application and so there should be some meta-rule stating how they should be applied: both, and if so which one first, or just one, and then which one.

[6] The prototype is implemented on top of the Tuscany [19] implementation of SCA, the Service Component Architecture [18] and is referred to as "SCA service" in Fig. 1.

This simple example indicates how complexity escalates when even straightforward concerns are combined. The challenge of determining policies for dealing with multiple non-functional concerns in concert is huge but we believe the idea of meta-rules will at least provide a framework in which these issues can be addressed.

6 Conclusions

We discussed how co-design of parallel/distributed computation structuring and autonomic management can facilitate the development of distributed systems by enforcing a separation of concerns at two levels. First, suitable abstractions may be provided to the application programmer, ensuring that he can concentrate on the core functional code and on specifying non-functional requirements as a SLA. In turn, many of the more challenging aspects of the distributed system development are left in the hands of the system programmer who is well placed to deal with these challenges. Second the system programmer is further aided by the separation of structure from (non-functional concern) management together with clear guidelines as to how the two should interface.

Preliminary results indicate that the approach is reasonable and feasible, both in the case COW/NOWs and of multi- many-core networks.

The proposed programming abstractions and co-design approach appears also to be suitable for implementing cloud programming environments, as they decouple programming effort from specific knowledge of the target architecture while, at the same time, preserving those positive aspects deriving from efficient implementation of both structuring and management patterns. Indeed, in the case of non-functional aspects in cloud computing, an approach similar to that proposed is essential, as the application programmer typically will have no possibility of directly managing such concerns.

References

1. Aldinucci, M., Campa, S., Danelutto, M., Vanneschi, M., Dazzi, P., Laforenza, D., Tonellotto, N., Kilpatrick, P.: Behavioural skeletons in GCM: autonomic management of grid components. In: Baz, D.E., Bourgeois, J., Spies, F. (eds.) Proc. of Intl. Euromicro PDP 2008: Parallel Distributed and network-based Processing, Toulouse, France, Feburary 2008, pp. 54–63. IEEE, Los Alamitos (2008)
2. Aldinucci, M., Coppola, M., Danelutto, M., Vanneschi, M., Zoccolo, C.: ASSIST as a research framework for high-performance grid programming environments. In: Cunha, J.C., Rana, O.F. (eds.) Grid Computing: Software environments and Tools, ch. 10, pp. 230–256. Springer, Heidelberg (2006)
3. Aldinucci, M., Danelutto, M., Teti, P.: An advanced environment supporting structured parallel programming in Java. Future Generation Computer Systems 19(5), 611–626 (2003)
4. Bacci, B., Danelutto, M., Orlando, S., Pelagatti, S., Vanneschi, M.: P^3L: a structured high level programming language and its structured support. Concurrency Practice and Experience 7(3), 225–255 (1995)

5. Caromel, D., Leyton, M.: Fine tuning algorithmic skeletons. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 72–81. Springer, Heidelberg (2007)
6. Cole, M.: Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming. Parallel Computing 30(3), 389–406 (2004)
7. Danelutto, M.: On skeletons and design patterns. In: Joubert, G.R., Murli, A., Peters, F.J., Vanneschi, M. (eds.) Parallel Computing: Advances and Current Issues (Proc. of Intl. ParCo 2001), Naples, Italy, pp. 425–432. Imperial College Press, London (2001)
8. Danelutto, M.: QoS in parallel programming through application managers. In: Proc. of Intl. Euromicro PDP: Parallel Distributed and network-based Processing, Lugano, Switzerland, pp. 282–289. IEEE, Los Alamitos (2005)
9. Danelutto, M., Zoppi, G.: Behavioural skeletons meeting services. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 146–153. Springer, Heidelberg (2008)
10. Grid 5000 home page (2008), http://www.grid5000.fr
11. GridCOMP Use case home page (2008), http://gridcomp.ercim.org/content/view/41/39/
12. JBoss rules home page (2008), http://www.jboss.com/products/rules
13. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1), 41–50 (2003)
14. Kuchen, H.: A skeleton library. In: Monien, B., Feldmann, R.L. (eds.) Euro-Par 2002. LNCS, vol. 2400, pp. 620–629. Springer, Heidelberg (2002)
15. Matsuzaki, K., Iwasaki, H., Emoto, K., Hu, Z.: A library of constructive skeletons for sequential style of parallel programming. In: InfoScale 2006: Proceedings of the 1st international conference on Scalable information systems. ACM, New York (2006)
16. Mattson, T.G., Sanders, B.A., Massingill, B.L.: Patterns for Parallel Programming. Addison-Wesley Professional, Reading (2005)
17. ProActive home page (2006), http://www-sop.inria.fr/oasis/proactive/
18. Service component architecture (2008), http://www.ibm.com/developerworks/library/specification/ws-sca/
19. Tuscany home page (2008), http://incubator.apache.org/tuscany/

Distributed Data Mining Tasks and Patterns as Services

Domenico Talia

University of Calabria, DEIS and ICAR-CNR, Via P. Bucci 41c,
87036 Rende, Italy
talia@deis.unical.it

Abstract. This paper discusses large-grain programming issues in data intensive applications designed for Grids and distributed infrastructures. We outline how Grid-based and service-oriented programming mechanisms can be developed as a collection of Grid/Web/Cloud services and investigate how they can be used to develop distributed data analysis tasks and knowledge discovery applications exploiting the SOA model. Then we discuss a strategy based on the use of services for the design of open distributed knowledge discovery tasks and applications on Grids and distributed systems. Some examples of frameworks developed according to this approach are outlined.

Keywords: Grid services, distributed programming abstractions, distributed data mining, knowledge discovery.

1 Introduction

Bigger and more complex problems must be solved today by using distributed computing technology and systems. Increasingly complex applications implemented on distributed systems like Grids, peer-to-peer (P2P) networks and Clouds are data bound (or data intensive). This means that they access and use large amounts of data that often are stored in distributed repositories or data centers.

Data sources available now and in a near future in digital formats are larger and larger and the number of applications in science and business that use them profitably are many and are increasing. This required the use of distributed computing infrastructures such as Grids, P2P and Cloud systems both to store/access them and process them in an efficient way by exploiting distributed programming and parallel programming techniques and tools.

The information stored in digital data archives is enormous and its size is still growing very rapidly. IDC estimated that in 2006 the humankind has created about 161 exabytes (161 billion gigabytes) of digital information and the production trend will be more than linear in the next years. Whereas until some decades ago the main problem was the shortage of information, the challenge now seems to be

- the very large volume of information to deal with and
- the associated complexity to process it and to extract significant and useful parts or summaries.

This results in large data availability that if will not be appropriately managed will become a data deluge that will not allow users to handle that massive amount of data and extract useful and understandable information and knowledge from it.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 415–422, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In fact, today the main problem is not storing data, but it is query, analyze, mine, and process large data sets. Techniques and development models coming from distributed programming, parallel computing, service oriented programming, and workflow design are vital to develop data intensive applications in high performance distributed computing infrastructures that are available today.

This paper discusses large-grain programming issues in data intensive applications designed for Grids and distributed infrastructures. We outline how Grid-based and service-oriented programming mechanisms can be developed as a collection of Grid/Web/Cloud services and investigate how they can be used to develop distributed data analysis tasks and knowledge discovery applications exploiting the SOA model. Then we discuss a strategy based on the use of services for the design of open distributed knowledge discovery tasks and applications on Grids and distributed systems.

2 Distributed Data Analysis Patterns

In conjunction with the data availability trend, we register today advancements in the area of distributed computing infrastructures that become more and more pervasive, dynamic, heterogeneous and large scale. In this new and evolving scenario, the development of data intensive applications must be high level with respect to the underlying computing and data management platforms. Therefore, is vital to design and implement programming abstractions and mechanisms that help designers to develop distributed applications on these processing infrastructures.

In designing abstractions and mechanisms for high level programming of *data intensive* applications and systems several issues should addressed to provide general solutions and avoid to miss important functionality and/or performance goals. Among the main issues that must be considered are:

- *Management of input data, internal data, and output data*. Mechanisms must be devised to program data input, transformations and output in data intensive applications. Here the main question is if the programming abstractions that today are included in programming tools are sufficient to handle with massive data management.
- *Dynamic data access*. The issue mentioned in the previous item is even more complex when data used in distributed applications are dynamic as it occurs in data streaming applications or in elastic computing environments where to accessible data can change in size, content and properties.
- *Data dependency*. In distributed systems that aggregate resources on demand or on their availability, formalisms capable to express dynamic data dependency could be of great help to programmers. Access to databases, file systems or Web repositories could change in time in recent and future distributed infrastructures. This requires to allow designers expressing in dependency of data in programming applications. Constructs and programming models that link operations (instructions, methods or services) to available data will help.
- *Dynamic task graphs/workflows*. In dynamic environments like Grids, P2P systems and Clouds, many applications could be programmed as dynamic

task graphs or dynamic workflows (e.g., service workflow) that adapt to the available resources or to the application requirements that can change in time (especially for long running applications). This research area is promising and need to be investigated also considering high level building blocks for programming task graphs and workflow in distributed systems. In such dynamic scenario, also data dependency can play a role.

- *Data parallelism vs task parallelism.* Abstractions for expressing parallelism and concurrency in data intensive applications is one of the key elements in designing high performance applications. When operations on data are independent, data parallel constructs can be effectively used and are to be preferred to task parallel patterns. However, we must be aware that in several data intensive application classes abstractions that are more complex than data parallel operations are needed. In distributed query executions, in parallel data mining and similar cases, task parallelism is needed to express complex and dynamic algorithms. Combination of both models must be considered in languages and environments that aim to be general and provide support for different programming approaches.
- *Parallel data mining and/or distributed data mining*. As today we are much more able to store and access data than analyze them, data mining algorithms and applications must be facilitated through the definition of parallel and/or distributed abstractions for programming data mining tasks on high performance infrastructures such as clusters, Grids, P2P systems and Clouds. The example of the MapReduce [1] pattern used to program highly parallel data mining and data analytics applications is significant in this context. Moreover, it should be considered if and how to integrate parallel constructs to be used in tightly coupled systems with distributed abstractions to be used for data mining in loosely coupled systems that are geographically distributed.
- *Programming level(s) for distributed mining operations/taks/patterns*. Abstractions for programming data mining algorithms can be defined at different levels and these levels influence operation grain size and complexity, abstraction, number of process/thread typically involved, communication model, etc. Figure 1 shows three levels with different programming models, languages, libraries and services. Each of them represent a class of abstractions that offer different mechanisms for programming data analysis algorithms. Some of them in several cases are not orthogonal and can be used in complex distributed applications where, for example, communication primitives, thread creation methods, master-slave patterns, should put together with Web services, mushups, and workflows.

After discussing some main issues in designing abstractions for programming distributed data intensive applications, in the remainder of the paper we focus on the study of a service based approach for providing abstractions for distributed data mining in service oriented distributed infrastructures.

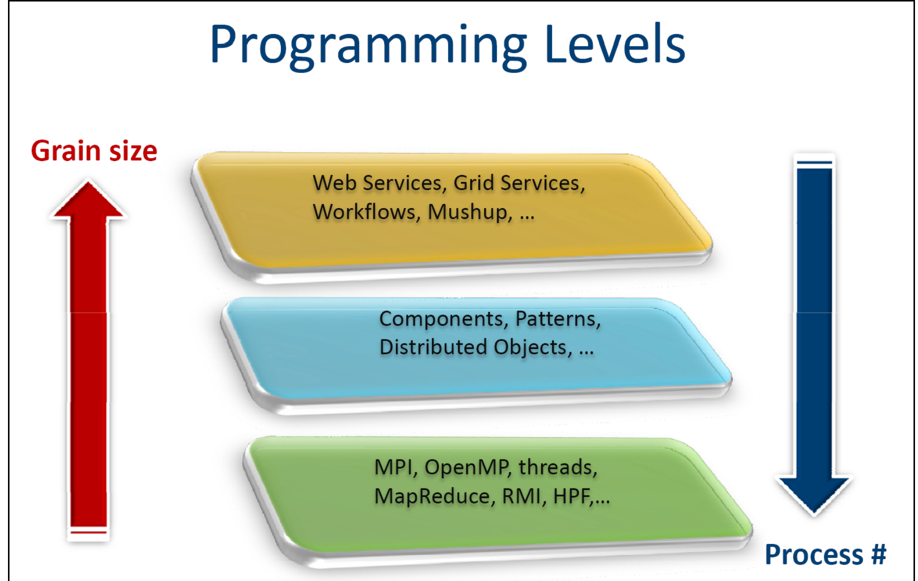


Fig. 1. Different programming levels that provide different abstractions for parallel and distributed programming

3 Grid/Web Services for Distributed Data Mining

Distributed infrastructures like Grids and Clouds extend the distributed and parallel computing paradigms allowing resource negotiation, dynamical allocation, heterogeneity, open protocols and services. As Grids and Clouds became well accepted computing infrastructures it is necessary to provide data mining services, algorithms, and applications [2].

Those services may help users to leverage capability of Grids, Clouds and, in general, service oriented Internet infrastructures, in supporting high-performance distributed computing for run their data mining tasks and applications. For example, by exploiting the SOA model and the Web Services Resource Framework (WSRF) in Grids it is possible to define basic services for supporting distributed data mining tasks and data analytics applications. These services can be also interoperable with other services developed with different technologies and also with legacy code that will be exposed as a service.

Those data analysis services can address all the aspects that must be considered in data mining and in knowledge discovery processes such as

- data selection and transport services,
- data querying services,
- data analysis services,
- knowledge models representation services, and
- visualization services.

According to this approach we can provide service oriented programming abstractions for distributed data mining that can be used at different levels form single operations on data to distributed data mining patterns and complete KDD processes run as service-based workflows on geographically remote sets of machines. Figure 2 summarizes four classes of data mining services that can be implemented in dynamic large scale distributed infrastructures.

Fig. 2. Four classes/levels of services that can be developed to facilitate the high level programming of distributed data mining tasks and KDD processes in Grids, Clouds or P2P networks. Services developed at one level can be used to implement services in the lower levels (in the figure).

It is worth to notice that services provided at one level can be used to implement services in other levels, thus, referring Figure 2, single step services can used to implement single data mining tasks or distributed data mining patterns. This incremental approach avoids the re-implementation of already available operations, tasks or patterns.

More significantly, we must point out that this collection of data mining services can constitute an *Open Framework for Service-based Data Mining* that allows developers to program distributed KDD processes as a composition of single and/or aggregated services available over a Grid or in a Cloud computing framework. Those services can exploit other basic Grid/Cloud/P2P services for data transfer, replica management, data integration and querying.

By exploiting this *Open Framework for Service-based Data Mining* in Grids, Clouds and dynamic distributed infrastructures it is possible to develop data mining services accessible every time and everywhere. This approach will result in

- Service-based distributed data mining applications;
- Data mining services for virtual organizations;
- Distributed data analysis services on demand;

Therefore, we could have a sort of knowledge discovery eco-system composed of a large numbers of decentralized data analysis services that will help users to face the availability of massive amounts of data both in business and science.

A question that could be raised after presenting the discussed approach is: Can be distributed data mining services considered programming abstractions? A quick response to this question could be: Apparently not, at least in a traditional programming approach. However, in our opinion the correct response should be: Yes, if we consider user and application requirements in handling data and in understanding what is useful in it. The approach can be a step towards distributed programming patterns for services in which we can have

- Basic services as simple operations;
- Complex services and their complex composition as libraries/patterns of operations;
- Service programming languages for composing them.

4 Some Service Frameworks for Distributed Data Mining

To validate our approach and experiment the design and use of distributed data mining services we recently developed a few systems. They are the Knowledge Grid [3], Weka4WS [4], Mobile Data Mining Grid Services [5], and Mining@home [6]. It is out of the scope of this paper to describe these systems; however, just to give the reader some info on them we report here about their main features.

The Knowledge Grid is a Grid service-based environment providing knowledge discovery services that can be used in high performance distributed applications. It includes high-level abstractions and a set of services by which users can integrate Grid resources to be used in each phase of a knowledge discovery process. The Knowledge Grid architecture is composed of two groups of services classified on the basis of their roles and functionalities. Indeed, two main aspects characterize a knowledge discovery process performed in accordance with the Knowledge Grid philosophy. The first is the management of data sources, data sets and tools to be used in the whole process. The second is concerned with the design and management of a knowledge flow that is the sequence of steps to be executed in order to perform a complete knowledge discovery process by exploiting the advantages coming from a Grid environment.

The goal of Weka4WS is to extend the Weka open source framework to support remote execution of the data mining algorithms in service-oriented Grid environments. To enable remote invocation, each data mining algorithm provided by the Weka library is exposed as a WSRF-compliant Web service which can be easily deployed on the available Grid nodes. Thus, Weka4WS also extends the Weka GUI to enable the invocation of the data mining algorithms that are exposed as Web services on remote machines.

Other than the two mentioned frameworks supporting the development of data mining applications on "wired" Grids, we implemented a mobile data mining system based on a wireless service oriented architecture. Here we refer to mobile data mining as the process of using mobile devices for running data mining applications involving remote computers and remote data. The availability of client programs on mobile

devices that can invoke the remote execution of data mining tasks and show the mining results is a significant added value for nomadic people and organizations. Those users need to perform analysis of data stored in repositories far away from the site where they work, thus mobile mining services allow them to generate knowledge regardless of their physical location. We implemented pervasive data mining of databases from mobile devices through the use of standard and WSRF-compliant Web services. By implementing mobile Web services, the system allows remote users to execute data mining tasks on a Grid or on the Internet from a mobile phone or a PDA and receive on those devices the results of a data analysis task. The mobile data mining Grid services have been implemented using the WSRF Java library provided by GT4 and a subset of the Weka library as data mining algorithms. The mobile client has been implemented by the Sun Java Wireless Toolkit, a widely adopted suite for the development of J2ME applications.

Finally, in the Mining@home work we aimed at exploring the opportunities offered by the volunteer computing paradigm for making feasible the execution of compute-intensive data mining jobs that have to explore very huge data sets. Mining@home introduces a novel data-intensive computing model, which is able to efficiently carry out mining tasks by adopting the volunteer computing paradigm. The network exploits caching techniques across a super-peer network to leverage the cost of spreading large amounts of data to all the computing peers.

5 Summary and Conclusion

New high performance parallel and distributed infrastructures allow us to attack new problems, but the efficient exploitation of their computing and storing power requires to solve more challenging problems.

New programming models and environments are required to design and implement efficient software systems and applications that can benefit of new dynamic, heterogeneous and pervasive infrastructures that are available today and those that will be available in the next years. In this paper we discussed requirements and features of distributed programming paradigms from the perspective of massive data analysis and focused on distributed data mining as a field where service oriented programming can help to compose complex applications and build knowledge discovery eco-systems constructed by a large numbers of decentralized data analysis services.

As data is becoming a big player, programming data analysis applications and services is a must in distributed infrastructures. New ways to efficiently compose different distributed models and paradigms are needed and relationships between different programming levels must be addressed.

In a long-term vision, pervasive collections of data analysis services and applications must be accessed and used as public utilities. Researchers and professionals must be ready for managing with this scenario and appropriate and efficient programming paradigms must be developed to support designers and programmers in their challenging task.

References

1. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Custers. CACM 51(1), 107–113 (2008)
2. Talia, D., Trunfio, P.: Service-Oriented Architectures for Distributed and Mobile Knowledge Discovery. In: Kargupta, H., Han, J., Yu, P., Motwani, R., Kumar, V. (eds.) Next Generation of Data Mining. CRC Press, Boca Raton (2008)
3. Congiusta, A., Talia, D., Trunfio, P.: Distributed data mining services leveraging WSRF. Future Generation Computer Systems 23(1), 34–41 (2007)
4. Talia, D., Trunfio, P., Verta, O.: The Weka4WS Framework for Distributed Data Mining in Service-Oriented Grids. Concurrency and Computation: Practice and Experience 20(16), 1933–1951 (2008)
5. Talia, D., Trunfio, P.: Mobile Data Mining on Small Devices Through Web Services. In: Yang, L., Waluyo, A., Ma, J., Tan, L., Srinivasan, B. (eds.) Mobile Intelligence: Mobile Computing and Computational Intelligence. John Wiley & Sons, Chichester (2008)
6. Barbalace, D., Lucchese, C., Mastroianni, C., Orlando, S., Talia, D.: Mining@home: Public Resource Computing for Distributed Data Mining. In: Priol, T., Vanneschi, M. (eds.) From Grids to Service and Pervasive Computing, pp. 217–227. Springer, Heidelberg (2008)

ProActive Parallel Suite: From Active Objects-Skeletons-Components to Environment and Deployment

Denis Caromel and Mario Leyton

INRIA Sophia Antipolis, Université de Nice Sophia Antipolis, CNRS - I3S
2004, Route des Lucioles, BP 93, F-06902 Sophia-Antipolis Cedex, France
First.Last@sophia.inria.fr

Abstract. The Proactive Parallel Suite offers multiple layers of abstraction for parallel and distributed applications which include both programming and the environment/deployment abstraction layers.

At the core of ProActive's programming abstractions are *active objects* with transparent futures and wait-by-necessity. Other abstractions offered by ProActive, such as *typed groups*, *algorithmic skeletons*, and *hierarchical distributed components* among others; are constructed on top of active objects. This pluralism of abstractions offers programmers a wide choice of expressiveness for coding parallel and distributed applications.

Additionally, an environment/deployment layer offers abstractions that simplify the interaction with the infrastructure. A deployment descriptor and a super-scheduler abstractions manage deployment of application on distributed resources, while the IC2D tool provides an abstraction to monitor debug and profile parallel and distributed applications.

1 Introduction

The relevance of parallel programming is evident. There has never been a point in time where we have had a dearer need for parallel programming abstractions to harness the power of increasingly complex parallel systems [40]. On one side large scale distributed-memory computing such as cluster and grid computing [29]; and on the other parallel shared-memory computing through new multi-core processors [7].

The difficulties of parallel programming have led to the development of many parallel programming models, each having its particular strengths. One thing which they have in common is that parallel programming models pursue a balance between abstractions (simplicity) and details (expressiveness). As applications increase in complexity, a single programming abstraction lacks expressiveness to adequately satisfy the whole application. Instead an approach where multiple abstractions are used for particular parts of the application is better suited. This papers describes a library providing such pluralism of programming models, the ProActive Parallel Suite.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 423–437, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The *ProActive Parallel Suite* is a 100% Java library, which aims at achieving seamless programming for concurrent, parallel, distributed, and mobile computing. It does not require any modification of the standard Java execution environment, nor does it make use of a special compiler, pre-processor, or modified virtual machine. Released under the GPL license, ProActive is a Java library for parallel, distributed, and concurrent computing; also featuring mobility and security in a uniform framework. With a reduced set of simple primitives, ProActive provides a comprehensive API which simplifies the programming of applications distributed on Local Area Networks (LAN), clusters, and Grids.

ProActive provides two levels of abstractions. First, a set of *programming model abstractions* such as: active objects, typed groups, algorithmic skeletons, distributed components, etc. The programming model abstractions are all implemented on top of the core programming abstraction, active objects, because they provide good properties such as determinism and orthogonality of future update policies [21]. Second, ProActive is also concerned with the complexity of deploying distributed applications. ProActive provides deployment descriptors which abstract low level information from the application source code. Users can deploy their applications on different infrastructures by providing the corresponding deployment descriptor, without changing the application code. Also, for more dynamic environments, ProActive supports batch like deployment of applications through an active object based scheduler. Additionally, ProActive provides a monitoring, debugging and profiling tool IC2D. Among others, IC2D provides a visual representation of an application's active objects and their communication.

This paper is organized as follows. Section 2 describes the related work. Section 3 describes the parallel programming abstractions in ProActive, starting with the active object model in Section 3.1, typed groups in Section 3.2, Calcium's algorithmic skeletons in Section 3.3, and the GCM hierarchical components in Section 3.4. Then Section 4 describes the Environment and Deployment abstractions. Finally Section 5 provides the conclusions and future work.

2 Related Work

ASSIST [2] is a programming environment which provides programmers with a structured coordination language. The coordination language can express parallel programs as an arbitrary graph of software modules. The graph describes how a set of modules interact with each other using a set of typed data streams. The modules can be sequential or parallel. Sequential modules can be written in C, C++, or Fortran; and parallel modules are programmed with a special ASSIST parallel module (*parmod*).

Condor [35] is a distributed computing system for batch processing. Condor provides job management, scheduling, resource monitoring and resource management. One of the key features of Condor is its matchmaking mechanism. Both jobs and resources describe their requirements using a ClassAd language, and the matchmaking determines if a resource is suitable for the execution of

a job. Jobs can be ordered using DAGMan to define the dependencies between jobs, and a Master-Worker system is available for parameter search applications. Condor also monitors the job's progress and informs of completion to the user.

GAT stands for *Grid Application Toolkit* [6,34], which defines a platform independent API to access resources and services. The API focuses, among others, on resource management (job submission and migration), and data management (file transfer, file access and communication pipes). A GAT Engine dispatches API calls to available services via *adaptors*. Adaptors are the interface between the GAT Engine and the third party services. They are analogous to providers in Java CoG. When a call to the API arrives, the GAT Engine executes suitable adaptors until one succeeds, or all fail, to perform the operation.

Globus Toolkit [30] is a rich set tools capable of interoperating to run applications on distributed and Grid infrastructures. These tools are concerned with deployment, data management, monitoring, and security among others. For deployment Globus provides GRAM which is the module responsible of the resource acquisition, configuration, executable staging, and program's execution. Data management is achieved by a set of tools, such as GridFTP used for data movement; but also others such as the Data Replication Service which handles replication of data.

Grid Superscalar [8] is an environment to program parallel applications for the Grid using imperative languages such as C++ and Perl. The program is specified as a set of tasks with input/output files in an interface description language. Grid Superscalar analyzes the dependencies between tasks and executes them sequentially or in parallel after having transferred the required data.

Java CoG Kit stands for *Java Commodity Grid Kit* [38], and provides services using simplified interfaces for lower level providers, in particular for the Globus Toolkit [31]. The Java CoG Kit's abstraction model follows a *provider* pattern, where abstract and generic concepts specified by programmers are translated into provider specific implementation entities. In the case of file transfer abstractions, file transfer operations are no different from other tasks, in the sense that a file transfer operation must be submitted for execution as a file-transfer-task [37,39].

SAGA stands for Simple API for Grid Applications, and has the same objective as GAT: to construct a uniform API for the development of Grid applications [32]. Indeed, SAGA is an API standardization effort within the Open Grid Forum (OGF). The SAGA API is concerned with functional features such as job submission and management, file input/output, replica management, remote procedure calls, etc; and non-functional features such as permissions, security, monitoring, etc.

Unicore [28] is a middleware oriented towards application Grid services, where services are setup on a pre-configured Grid environment. Remote clients submit jobs to the Unicore's Grid gateway, which chooses suitable resources to run the jobs. A job is composed of one or more typed tasks. Each tasks triggers the execution of a predefined Grid service, in accordance with the type of the task. Tasks are arranged using a workflow, and can be executed in parallel or not. All tasks belonging to the same job share a jobspace file system. Besides the

workflow, the job description also specifies which files must be imported into the jobspace before the execution of the job, and which files must be exported after the job is finished.

3 Parallel Programming Abstractions

ProActive provides several programming abstractions. This section describes only the following ones: active objects, typed groups, algorithmic skeletons, and components; which we believe provide a good overview of ProActive's pluralism of abstractions. Readers interested on some other specific programming model in ProActive, such as the Branch & Bound [19], Master-Slave, or Monte Carlo [16] APIs should refer to the ProActive documentation for further details [33].

3.1 Active Objects with Transparent Futures

At ProActive's core lies a uniform *active object* programming model abstraction. As shown in Figure 1, active objects are remotely accessible via method invocations, which are automatically stored in a queue of pending requests. Each active object has its own thread of control and is granted the ability to decide in which order the incoming method calls are served. Method calls on active objects are asynchronous with automatic synchronization. This is achieved using transparent *future objects* as a result of remote methods calls, and synchronization is handled by a mechanism known as *wait-by-necessity* [18].

Active objects are instantiated using the ProActive API, as shown in Listing 1, by specifying the class of the root object, the instantiation parameters, and an optional location node. Invoking the method `foo()` on `b` returns a future of

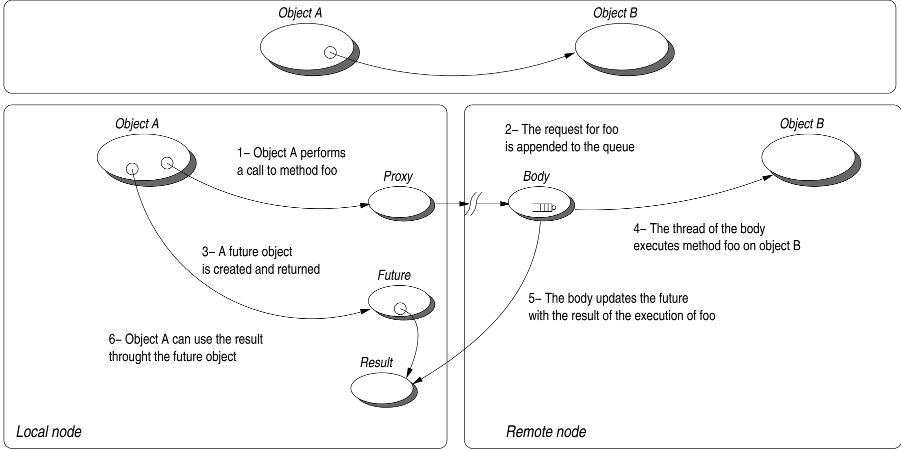


Fig. 1. Execution of a remote method call

```
Object[] params= ...; //Constructor parameters
// instantiate active object of class B on a remote node
B b = (B) ProActive.newActive("B", params, node);

// use active object as any object of type B
R r = b.foo();
...

// possible wait-by-necessity
System.out.println(r.printResult());
```

Listing 1. Active Object instantiation and method invocation

type `R`. Where `R` is the return type of the method foo, not a wrapper type. The computation can continue until a wait-by-necessity is reached. The thread accessing the future will be blocked only if the result is not yet available when it is actually required.

Active objects may migrate from any Java Virtual Machine (JVM) to any other using the provided *migration* mechanism. An active object with its pending requests (method calls), futures, and passive (mandatory non-shared) objects can migrate from JVM to JVM through the `migrateTo(...)` primitive. The migration can be initiated from outside the active object, but it is the responsibility of the active object to execute the migration, this is known as *weak migration*. Automatic and transparent forwarding of requests and replies provide location transparency, as remote references toward *active mobile objects* remain valid.

ProActive uses by default the RMI Java standard library as a portable communication layer, supporting the following communication protocols: `RMI`, `HTTP`, `Jini`, `RMI/SSH`, and `Ibis` [36].

3.2 Typed Groups

An extension of the active object abstraction corresponds to the *typed group communication* model [9]. Group communication is an important feature for high-performance and Grid computing, for which MPI is generally the only available coordination model [10]. Group communication allows triggering method calls on a distributed group of *active objects* with compatible type, dynamically generating a group of results. It has been shown in [9] that this group communication mechanism, plus a few synchronization operations (WaitAll, WaitOne, etc.), provides similar patterns for collective operations such as those available in MPI, but in a language centric approach [10].

The typed group communication mechanism [9] is built upon the ProActive elementary mechanism for asynchronous remote method invocation with automatic futures. The group mechanism must be thought of as a replication of more than one (say N) ProActive remote method invocations towards N active objects. Of course, the aim is to incorporate some optimizations into the group mechanism implementation, in such a way as to achieve better performances than a sequential achievement of N individual ProActive remote method calls. In this

```
Object[][] paramsArray = {{...},{...},...}; Node[] nodes =
{...,...,... }; A ag = (A) ProActiveGroup.newActiveGroup("A",
paramsArray, nodes); ... ag.foo(...); // A group communication

// A method call on a typed group
V vg = ag.bar();
// To wait and capture the first returned member of vg
V v = (V) ProActiveGroup.waitAndGetOne(vg);
// To wait all the members of vg are arrived
ProActiveGroup.waitAll(vg);
```

Listing 2. Typed Group Communications

way, the mechanism is a generalization of the remote method call mechanism of ProActive.

The availability of such a group communication mechanism simplifies the programming of applications with similar activities running in parallel. Indeed, from the programming point of view, using a group of active objects of the same type, subsequently called a typed group, takes exactly the same form as using only one active object of this type. This is possible due to the fact that the ProActive library is built upon reification techniques.

Listing 2 shows an example using typed group communication. The creation of a group is analogous to the creation of an active object but using the `newActiveGroup` primitive. A group communication call is transparent and the result is stored in a future. The API allows for several utility methods like `waitAndGetOne` which waits for a single result from the group, and `waitAll` which waits for all results.

3.3 Algorithmic Skeletons

Algorithmic skeletons (*skeletons* for short) are a high level programming model for parallel and distributed computing, introduced by Cole in [24]. Skeletons take advantage of common programming patterns to hide the complexity of parallel and distributed applications. Starting from a basic set of patterns (skeletons), more complex patterns can be built by nesting the basic ones. All the parallelization and distribution aspects are implicitly defined by the composed skeletal structure.

As a skeleton framework we use Calcium [20,22,23], which is greatly inspired on Lithium [3,4,5,27] and its successor Muskel [26]. Calcium is written in Java and is provided as a library. The Calcium framework is capable of evaluating the same skeleton program on different execution environments. Currently it supports parallel environments using threads, distributed environments using ProActive's active objects, and Grid like environments using the ProActive Scheduler (see Section 4.2).

In Calcium, skeletons are provided as a Java library. The library can nest task and data parallel skeleton in the following way:

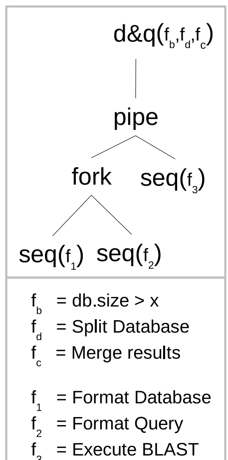
$$\triangle ::= \text{seq}(f_e) \mid \text{farm}(\triangle) \mid \text{pipe}(\triangle_1, \triangle_2) \mid \text{while}(f_b, \triangle) \mid$$
$$\text{if}(f_b, \triangle_{true}, \triangle_{false}) \mid \text{for}(i, \triangle) \mid \text{map}(f_d, \triangle, f_c) \mid$$
$$\text{fork}(f_d, \{\triangle_i\}, f_c) \mid \text{d\&c}(f_d, f_b, \triangle, f_c)$$

Each skeleton represents a different pattern of parallel computation. All the communication details are implicit for each pattern, hidden away from the programmer, and are classified in two types: task parallel or data parallel. The task parallel skeletons are: $farm$ for task replication; *pipe* for staged computation; *seq* for wrapping execution functions; if for conditional branching; and $while/for$ for iteration. The data parallel skeletons are: *map* for single instruction multiple data; $fork$ for multiple instruction multiple data; and *d&c* for divide and conquer.

The nested skeleton pattern ($\triangle$) relies on sequential blocks of the application. These blocks provide the business logic and transform a general skeleton pattern into a specific application. We denominate these blocks *muscles*, as they provide the *real* (non-parallel) functionality of the application. In Calcium, muscles come in four flavors:

Execution	$f_e : P \to R$
Division	$f_d : P \to \{R\}$
Conquer	$f_c : \{P\} \to R$
Condition	$f_b : P \to \text{boolean}$

Where P is the parameter type, R the result type, and $\{X\}$ a list of parameters or results of type X.




```
//Initialization
Skeleton<BlastParams,File> blast = ...;
Environment env = new ProActiveEnv(...);
Calcium calcium = new Calcium(env);
Stream<BlastParams,File> stream =
        calcium.getStream(blast);

//Input Parameters
stream.input(new BlastParams("/home/query.1"));
stream.input(new BlastParams("/home/query.2"));

//...

//Output Results
File alignment1 = stream.getResult();
File alignment2 = stream.getResult();
```

Fig. 2. BLAST Skeleton Program

For the skeleton language, muscles are black boxes invoked during the computation of the skeleton program. Multiple muscles may be executed either sequentially or in parallel with respect to each other, in accordance with the defined $\triangle$. The result of a muscle is passed as a parameter to other muscle(s). When no further muscles need to be executed, the final result is delivered to the user.

Figure 2 shows an example. BLAST [15] corresponds to Basic Local Alignment Search Tool. It is a popular tool used in bioinformatics to perform sequence alignment of DNA and proteins. In short, BLAST reads a query file and performs an alignment of this query against a database file. The results of the alignment are then stored in an output file. A BLAST parallelization using skeleton programming is shown in the figure. The strategy is to divide the database until a suitable size is reached and then merge the results of the BLAST alignment. The code shown in the figure represents the usage API. An initialization phase defines the skeleton program (portrayed graphically in the example), and then instantiates Calcium with a specific environment. The skeleton program is then associated with a stream which is used to input parameters and collect the results.

3.4 Grid Component Model (GCM)

The Grid Component Model (GCM) [25] abstraction extends Fractal [17] for distributed and Grid computing [12]. As in Fractal, GCM allows for hierarchical composition, separation of functional and non-functional interfaces; but also considers deployment, collective communications [14], and autonomic behavior [1] among others.

ProActive's GCM implementation is built upon the active object model. Each component is implemented with an active object and (non-)functional requests are served from the active object queue. Invocations on component interfaces inherit asynchronism from the remote method calls of active objects. The result of an invocation is also a transparent future which can be passed to other components.

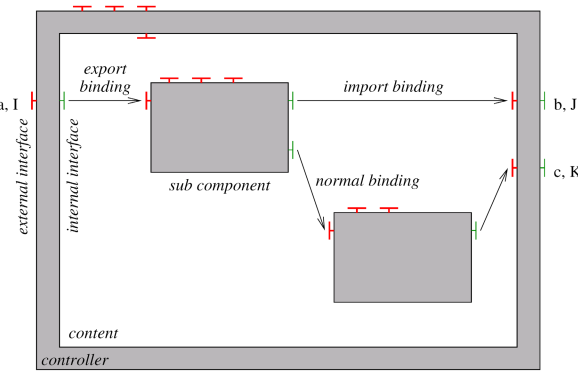


Fig. 3. Fractal based Grid Component Model

Figure 3 shows graphical example of a GCM component. External interfaces provide interaction with the environment while internal interfaces are binded with sub-components. The interface on the sides represent server (left) and client (right), while interfaces on the top correspond to non-functional services: life cycle, reconfiguration, etc.

Among others, GCM extends Fractal by providing multicast and gathercast interfaces [14]. Multicast interfaces provide abstractions for one-to-many communications, by transforming a single invocation into a list of invocations. The transformation is customizable (broadcast, split, etc), and the result of such invocation is a list of result or its reduction. The symmetrical interfaces are gathercast which provide abstractions for many-to-one communications by transforming a list of invocations into a single invocation. Gathercast interfaces can coordinate the invocation which is automatically redistributed to the invoking components.

4 Environment and Deployment Abstractions

4.1 Deployment and Scheduling

Descriptor Based. The deployment of distributed applications is commonly done manually through the use of remote shells for launching the various virtual machines or daemons on remote computers and clusters. In heterogeneous infrastructure the deployment complexity increases thus making the deployment task central and harder to perform by the application.

To address this issue, ProActive provides a *deployment descriptor* abstraction [11], which allows the deployment of applications on heterogeneous sites

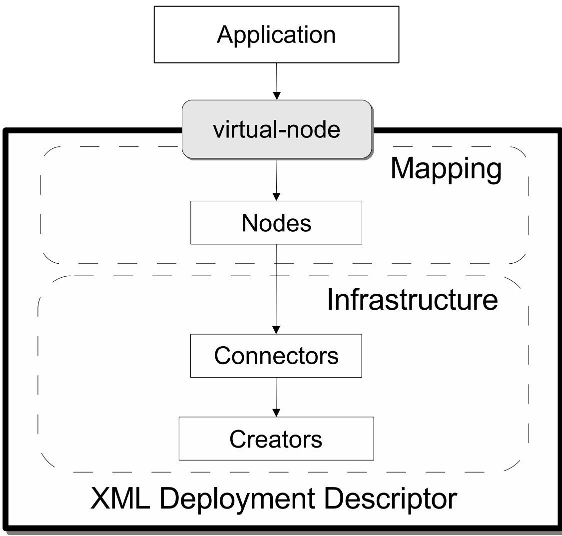


Fig. 4. Deployment Descriptor Layout

without changing the application's source code. All infrastructure information related with the deployment of applications on the infrastructure is described in a *deployment descriptor* (XML). Thus, eliminating references inside the code to: machine names, resource acquisition protocols (local, rsh, ssh, lsf, globus-gram, unicore, pbs, lsf, nordugrid-arc, etc...), and communication/lookup protocols (rmi, jini, http, etc...).

The deployment descriptor's architecture is shown in Figure 4. The infrastructure section contains the information necessary for acquiring remote resources. Once acquired, ProActive *nodes* are instantiated on the remote resources. The nodes are then linked with the application code via a `virtual-node` abstraction. In the application's code, a `virtual-node` name corresponds to a reference on the nodes that will be acquired during the deployment. While, on the deployment descriptor, the `virtual-node` corresponds to a set of deployment operations that will yield resources with instantiated nodes.

Consequently, the parsing of the deployment descriptor, and associated deployment operations, are triggered from the application code by calling one single method of the ProActive library. The deployment can be configured by changing the

$$\text{application} \rightarrow \texttt{virtual-node} \rightarrow \text{nodes}$$

mapping, to run the application on a different infrastructure, without modifying a single line of code in the application.

4.2 Scheduler

Batch schedulers provide an abstraction of resources to clients. Clients submit tasks and the scheduler is in charge of executing them on available resources. Thus, a scheduler allows several clients to share a same pool of resources. In this section, we present a super-scheduler (*scheduler* for short), capable of federating other schedulers. Clients can interact with the scheduler through different mechanism: command-line, API, GUI, and description files. In addition to the super-scheduler, we describe a *resource manager*, which is in charge of acquiring and managing resources.

The Scheduler is the central entity with which clients interact using a remote Java API, or by submitting a `Job` Description. A `Job` describes the batch process to be executed. The description specifies the code, which can be in Java by extending the `Executable` interface or any native executable; required data files; and a script for validating resources.

Currently three kinds of jobs are supported: in **Task Flow** Jobs, clients describe the flow and dependencies of tasks to execute; in **Parameter Sweeping** a single task is executed in parallel with multiple data; and in **ProActive Applications** clients submit a regular ProActive distributed application.

The scheduler also supports customized allocation policies, and provides a FIFO policy by default. Basic non-functional concerns such as security and fault-tolerance are handled both at the ProActive middleware and scheduler levels.

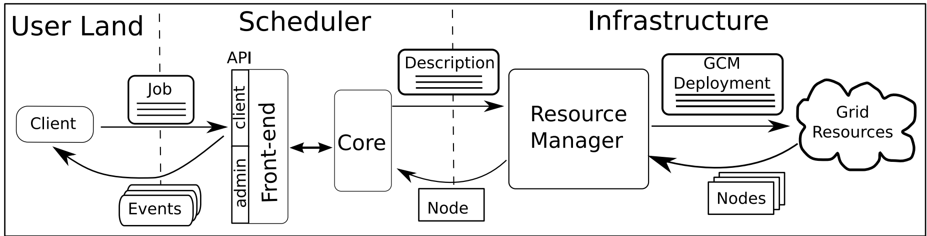


Fig. 5. Scheduler Global Overview

Finally, the management, deployment, and selection of resources is handled by a second entity, named the *resource manager*. Figure 5 shows a global overview of the whole system.

The Resource Manager (RM) is responsible for acquiring and managing resources. The RM is built on top of the deployment descriptors which provide an abstraction of how resources can be acquired.

The static acquisition of resources is handled by deployment descriptors, while the dynamic management of these resources is done by the RM. A scheduler can ask resources from a RM, and the RM will deliver a resource through a node abstraction. Once the scheduler no longer requires a node, it is returned to RM for cleaning, pooling or releasing.

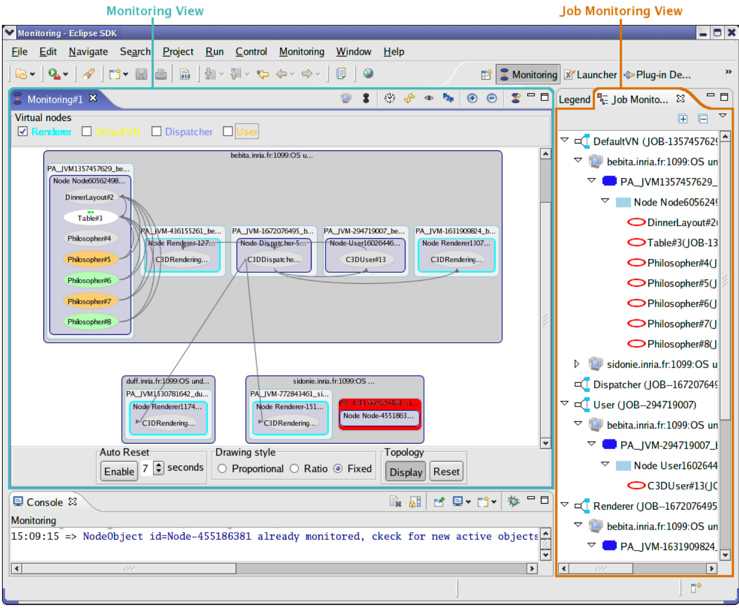


Fig. 6. IC2D Example Snapshot

The scheduler can also request specific resources, that fulfill some requirements in order to execute a particular task. Requirements can be verified with a script attached to the task, the RM uses this script to test resources. A successful execution of this script on a given resource validates the node.

4.3 IC2D Monitoring, Debugging, and Profiling

Graphical visualization and monitoring of any ongoing ProActive applications is possible with *IC2D* (Interactive Control and Debugging of Distribution) tool [13]. IC2D provides a graphical representation of hosts, java virtual machines, nodes, active objects with their queue of requests, and messages as shown in Figure 6. In the figure, outermost squares represent hosts while inner squares correspond to java virtual machines and nodes. The ellipsis correspond to active objects and the queue of pending requests is represented by dots inside active objects. Communications between active objects are shown by lines. Additionally, IC2D allows the monitoring of migrations, which can also be triggered through IC2D with a drag-and-drop.

When interfaced with Timit a profile of the application can be generated. The profile contains information such as time spent sending/waiting for requests, a timeline of activity for each active object, memory usage, and thread usage among others.

5 Conclusions

The ProActive Parallel Suite offers a variety of abstractions to ease the programming and execution of parallel and distributed applications. The programming abstraction layer is based on an active object model with transparent first class futures and wait-by-necessity. On top of this programming model, other abstractions are provided such as typed groups, Calcium's algorithmic skeletons, and GCM components among others. ProActive's programming models pluralism allows programmers to choose the most adequate abstractions for their application.

ProActive also provides an environment/deployment abstraction. The deployment process is simplified with deployment descriptors or a scheduler for more dynamic environments. Additionally, the IC2D tool provides abstract to monitor, debug, and profile parallel and distributed applications.

The current and future work of ProActive has many directions. For example the deployment mechanism is currently undergoing a complete re-write and extension. The new deployment labeled *GCM Deployment*, will harness all the experience gathered in the deployment of ProActive applications. Besides a simplified an easier description of infrastructure resources, the GCM Deployment considers an application side descriptor.

At the programming abstraction level, for active objects we are currently working on providing advanced update policies. For GCM components we are currently working on better MPI like collective communications, integration

with webservices, tailored monitoring, reconfiguration, and compositional non-functional aspects.

References

1. Aldinucci, M., Campa, S., Danelutto, M., Vanneschi, M., Kilpatrick, P., Dazzi, P., Laforenza, D., Tonellotto, N.: Behavioural skeletons in gcm: Autonomic management of grid components. In: PDP 2008: Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), Washington, DC, USA, pp. 54–63. IEEE Computer Society, Los Alamitos (2008)
2. Aldinucci, M., Coppola, M., Danelutto, M., Tonellotto, N., Vanneschi, M., Zoccolo, C.: High level grid programming with ASSIST. Computational Methods in Science and Technology 12(1), 21–32 (2006)
3. Aldinucci, M., Danelutto, M.: Stream parallel skeleton optimization. In: Proc. of PDCS: Intl. Conference on Parallel and Distributed Computing and Systems, Cambridge, Massachusetts, USA, November 1999, pp. 955–962. IASTED, ACTA press (1999)
4. Aldinucci, M., Danelutto, M., Dünnweber, J.: Optimization techniques for implementing parallel skeletons in grid environments. In: Gorlatch, S. (ed.) Proc. of CMPP: Intl. Workshop on Constructive Methods for Parallel Programming, Stirling, Scotland, UK, July 2004, pp. 35–47. Universität Münster, Germany (2004)
5. Aldinucci, M., Danelutto, M., Teti, P.: An advanced environment supporting structured parallel programming in Java. Future Generation Computer Systems 19(5), 611–626 (2003)
6. Allen, G., Davis, K., Goodale, T., Hutanu, A., Kaiser, H., Kielmann, T., Merzky, A., van Nieuwpoort, R.V., Reinefeld, A., Schintke, F., Schott, T., Seidel, E., Ullmer, B.: The grid application toolkit: Towards generic and easy application programming interfaces for the grid. In: Proceedings of the IEEE, March 2005, vol. 93, pp. 534–550 (2005)
7. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W., Yelick, K.A.: The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (December 2006)
8. Badia, R.M., Labarta, J., Sirvent, R., Perez, J.M., Cela, J.M., Grima, R.: Programming grid applications with grid superscalar. Journal of Grid Computing 1(2), 151–170 (2003)
9. Baduel, L., Baude, F., Caromel, D.: Efficient, Flexible, and Typed Group Communications in Java. In: Joint ACM Java Grande - ISCOPE 2002 Conference, Seattle, pp. 28–36. ACM Press, New York (2002) ISBN 1-58113-559-8
10. Baduel, L., Baude, F., Caromel, D.: Object-Oriented SPMD. In: Proceedings of Cluster Computing and Grid, Cardiff, United Kingdom (May 2005)
11. Baude, F., Caromel, D., Mestre, L., Huet, F., Vayssière, J.: Interactive and descriptor-based deployment of object-oriented grid applications. In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 2002, pp. 93–102. IEEE Computer Society, Los Alamitos (2002)

12. Baude, F., Caromel, D., Morel, M.: From distributed objects to hierarchical grid components. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 1226–1242. Springer, Heidelberg (2003)
13. Baude, F., Bergel, A., Caromel, D., Huet, F., Nano, O., Vayssière, J.: IC2D: Interactive Control & Debug of Distribution. In: Grappes 2001 (May 2001) (invited talk), `http://www.univ-ubs.fr/valoria/grappes2001`
14. Baude, F., Caromel, D., Henrio, L., Morel, M.: Collective interfaces for distributed components. In: CCGRID 2007: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, pp. 599–610. IEEE Computer Society, Los Alamitos (2007)
15. BLAST. Basic local alignment search tool, `http://www.ncbi.nlm.nih.gov/blast/`
16. Bossy, M., Baude, F., Doan, V.D., Gaikwad, A., Stokes-Rees, I.: Parallel pricing algorithms for multi–dimensional bermudan/american options using monte carlo methods. In: CoRR, abs/0805.1827 (2008)
17. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.-B.: The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. Software, Practice & Experience 36(11-12), 1257–1284 (2006)
18. Caromel, D.: Toward a method of object-oriented concurrent programming. Communications of the ACM 36(9), 90–102 (1993)
19. Caromel, D., di Costanzo, A., Baduel, L., Matsuoka, S.: Grid'BnB: A Parallel Branch & Bound Framework for Grids. In: Aluru, S., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2007. LNCS, vol. 4873, pp. 566–579. Springer, Heidelberg (2007)
20. Caromel, D., Henrio, L., Leyton, M.: Type safe algorithmic skeletons. In: Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-based Processing, Toulouse, France, pp. 45–53. IEEE CS Press, Los Alamitos (2008)
21. Caromel, D., Henrio, L., Serpette, B.: Asynchronous and deterministic objects. In: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 123–134. ACM Press, New York (2004)
22. Caromel, D., Leyton, M.: Fine tuning algorithmic skeletons. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 72–81. Springer, Heidelberg (2007)
23. Caromel, D., Leyton, M.: A transparent non-invasive file data model for algorithmic skeletons. In: 22nd International Parallel and Distributed Processing Symposium (IPDPS), Miami, USA, pp. 1–8. IEEE Computer Society, Los Alamitos (2008)
24. Cole, M.: Algorithmic skeletons: structured management of parallel computation. MIT Press, Cambridge (1991)
25. CoreGRID, Programming Model Institute. Basic features of the grid component model (assessed). Technical report, Deliverable D.PM.04 (2006), `http://www.coregrid.net/mambo/images/stories/Deliverables/d.pm.04.pdf`
26. Danelutto, M., Dazzi, P.: Joint structured/unstructured parallelism exploitation in Muskel. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3992, pp. 937–944. Springer, Heidelberg (2006)
27. Danelutto, M., Teti, P.: Lithium: A structured parallel programming environment in java. In: Sloot, P.M.A., Tan, C.J.K., Dongarra, J., Hoekstra, A.G. (eds.) ICCS-ComputSci 2002. LNCS, vol. 2330, pp. 844–853. Springer, Heidelberg (2002)

28. Erwin, D.W., Snelling, D.F.: Unicore: A grid computing environment. In: Sakellariou, R., Keane, J.A., Gurd, J.R., Freeman, L. (eds.) Euro-Par 2001. LNCS, vol. 2150, pp. 825–834. Springer, Heidelberg (2001)
29. Foster, I., Kesselman, C.: The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc., San Francisco (1999)
30. Foster, I.T.: Globus toolkit version 4: Software for service-oriented systems. In: Jin, H., Reed, D.A., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 2–13. Springer, Heidelberg (2005)
31. Globus, `http://www.globus.org`
32. Kaiser, H., Merzky, A., Hirmer, S., Allen, G., Seidel, E.: The saga c++ reference implementation: a milestone toward new high-level grid applications. In: SC 2006: Proceedings of the, ACM/IEEE conference on Supercomputing, p. 184. ACM, New York (2006)
33. ProActive, `http://proactive.objectweb.org`
34. Seidel, E., Allen, G., Merzky, A., Nabrzyski, J.: Gridlab: A grid application toolkit and testbed. Future Generation Computer Systems 18, 1143–1153 (2002)
35. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience: Research articles. Concurrency and Computation: Practice & Experience 17(2-4), 323–356 (2005)
36. van Nieuwpoort, R., Maassen, J., Wrzesinska, G., Hofman, R., Jacobs, C., Kielmann, T., Bal, H.: Ibis: a flexible and efficient java-based grid programming environment. Concurrency - Practice and Experience 17(7-8), 1079–1107 (2005)
37. von Laszewski, G., Alunkal, B., Gawor, J., Madhuri, R., Plaszczak, P., Xian-He, S.: A File Transfer Component for Grids. In: Arabnia, H.R., Mun, Y. (eds.) Proceedings of the International Conferenece on Parallel and Distributed Processing Techniques and Applications, vol. 1, pp. 24–30. CSREA Press (2003)
38. von Laszewski, G., Foster, I., Gawor, J., Lane, P.: A Java commodity grid kit. Concurrency and Computation: Practice and Experience 13(8–9), 645–662 (2001)
39. von Laszewski, G., Gawor, J., Plaszczak, P., Hategan, M., Amin, K., Madduri, R., Gose, S.: An overview of grid file transfer patterns and their implementation in the java cog kit. Neural, Parallel Sci. Comput. 12(3), 329–352 (2004)
40. Yelick, K.: Keynote: Programming models for petascale to exascale. In: 22nd International Parallel and Distributed Processing Symposium (IPDPS), Miami, USA, March 2008, pp. 1–1. IEEE Computer Society, Los Alamitos (2008)

On Abstractions of Software Component Models for Scientific Applications*

Julien Bigot[1], Hinde Lilia Bouziane[1], Christian Pérez[1], and Thierry Priol[2]

[1] INRIA/LIP, ENS Lyon, 46 allée d'Italie, F-69364 Lyon Cedex, France
[2] INRIA/IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, France
{Julien.Bigot,Hinde.Bouziane,Christian.Perez,
Thierry.Priol}@inria.fr

Abstract. In the seek of more computing power, two sources of complexity are to face. On one hand, it is possible to aggregate a large amount of computing power (and storage) at the price of very complex resources such as grids. On the other hand, such available computing power allows to imagine more complex applications such as code coupling applications to achieve more realistic simulations. Component models appear as a solid foundation to handle simultaneously both sources of complexity. However, component models need to provide adequate abstractions to offer a simple programming model while enabling high performance on any kind of resources. This paper reviews several abstractions dedicated for scientific applications: data sharing between components, master-worker relationships, parallel to parallel component communications and collective communications among components.

1 Introduction

In order to better simulate the reality, applications are always looking for more computing power – as well as storage space. A huge computing power is available but at the price of a huge complexity to harness it such as in grids, with recently multi-core multi-CPU nodes. Moreover, to make use of such a computing power, applications turn out to get more and more complex. For example, code coupling applications aim to compose several (complex) codes coming from independent teams.

Hence, there is a clear need of a programming model that can handle simultaneously both sources of complexity so as to hide the infrastructure complexity (heterogeneity, volatility, etc.) while providing a simple model to efficiently build applications from several pieces of code.

Software component technology appears to provide an interesting foundation to reach such a goal. Software components aim at handling code reuse [1] and distributed software component models such as CCM [2] or SCA [3] have already dealt with code and resource heterogeneity. Generic and hierarchical component models such as FRACTAL [4] provide a foundation where more specific component models such as GCM can be defined [5]. Dedicated high performance models have also been proposed such as CCA [6].

* This work was supported by the CoreGRID European Network of Excellence and by the French National Agency for Research project LEGO (ANR-05-CIGC-11).

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 438–449, 2009.
© Springer-Verlag Berlin Heidelberg 2009

However, a common limitation of these models is that their programming model is very close to their execution model. It is very annoying for application portability as the programmer has to know the infrastructure architecture when designing its application. Moreover, it is not any longer satisfactory as more and more resources are volatile. Therefore, several researches have been conducted to increase the abstraction level of component models. For example, behavioral skeletons have been recently added to GCM [7]. Such an improvement of the abstraction level of component models a) improves the productivity by avoiding duplicating the effort for recurring composition patterns; b) reduces the complexity by embedding expertise in a "simple" concept; and c) hides the implementation details and thus enables resource independent composition. Hence, the challenge is to simplify application development while keeping high performance.

This paper presents four abstractions – namely data sharing, master-worker paradigm, parallel component and collective communications. These abstractions take place at the different levels of a component model: the port level (data sharing and collective communication), the component level (parallel component), and the assembly level (master-worker paradigm and collective communications). Our aim is to show that it is possible to simplify application development while keeping high performance.

The remaining of this paper discusses the four abstractions. Section 2 deals with data sharing between components. The support of the master-worker paradigm in component models is studied in Section 3. Section 4 is devoted to parallel components and Section 5 focuses on collective communications between components. Section 6 concludes the paper.

2 Data Sharing

2.1 Presentation

The *shared memory* paradigm is an attractive programming paradigm that allows data to be shared by multiple concurrent entities. Its advantage relies on the ease of programming: multiple entities (threads, processes, etc) can concurrently read/write data in a global space without any need to explicitly handle data localization, transfer and persistence. This concept has been successfully applied in several contexts: (1) multithreading within the same process, (2) data segment sharing among multiple processes running on the same host, (3) global data sharing across a cluster of workstations through Distributed Shared Memory (DSM) systems, and (4) grid data sharing services such as JUXMEM [8] in grid environments. Such a service transparently manages data localization and persistence in a dynamic, large-scale, distributed environment. A common objective of underlying data sharing systems is to hide the complexity of data sharing management. They also deal with non-functional concerns related to the nature of targeted execution resources and processes placement. Resources volatility and performance are examples of such specificities.

Sharing data among multiple computation entities is appropriate for some applications where data structures are complex and access patterns are irregular. Such applications may be component based applications. Hence, it is interesting to bring the benefits of the shared memory paradigm into component models.

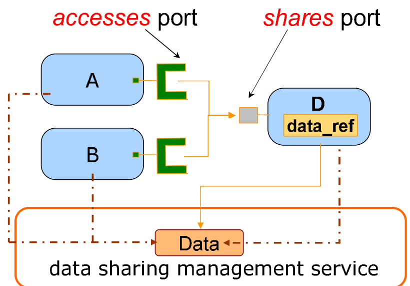


Fig. 1. Overview of data shared ports

```
interface AccessPort {
    float* get_pointer();
    long get_size();
    // Synchronization primitives
    void  acquire();
    void  acquire_read();
    void  release();
}
```

Fig. 2. Example of data port API accessing an array of floats

2.2 Limitations with Existing Component Models

In a classical way, interactions between components are done through well-defined ports. In fact, in existing component models, ports only enable explicit data transfer where the data is part of an exchanged message. Consequently, it is not easy to share data between components. When several components want to modify a same data, the functional code of a component should deal with data persistence, data consistency and fault tolerance issues. This therefore leads to an increased code complexity.

2.3 Data Shared Port Model

In [9], we proposed an additional family of ports named *data shared ports* (Figure 1). A data port logically attaches a shared data to a component. It can be of two kinds: a *shares* port to give an access to a shared data and an *accesses* port to enable a component to access a data exported through a *shares* port. These ports rely on a *transparent data access model*.

The proposed model provides a user view divided in two parts: an internal view that allows the implementation of a component to access a data and an external view that allows a component to share a data. In the internal user view, an interface named *AccessPort*, is implicitly associated to a data port. This interface is shown in Figure 2. It is available through *accesses* ports as well as *shares* ports. It allows a component with a *shares* port to also access the associated data. The *AccessPort* interface provides `get_pointer` and `get_size` operations to respectively retrieve a pointer to the shared data and its size. It also provides synchronization primitives, like `acquire` and `release`. The `acquire_read` primitive sets a lock in a read-only mode so that multiple readers can simultaneously access the shared data, whereas `acquire` sets a lock in an exclusive mode.

In the external view, a component which aims to access a data through an *accesses* port should have this port connected to a *shares* one. Such a connection implies passing the reference of the shared data from the shares component to the accesses component. It is assumed that the shared data is previously allocated and associated to the *shares* port. That is done through a dedicated interface provided only on the *shares* port side.

The global user view of data ports allows data to be shared between components without worrying about the mechanism used to share the data. Such a mechanism can

be a memory shared between components collocated within a same process, a shared memory segment for components in two different processes but deployed on the same host, a DSM for a cluster or a grid data-sharing service like JUXMEM for a grid. The choice of a particular mechanism is expected to be done by the execution/deployment framework once execution resources and the placement of components are known. This choice determines which implementation of data port interfaces is to be used. The ability to use several implementations does not require to modify the user code of components.

2.4 Implementation

The proposal was projected on CCM [9] and CCA [10]. The projection is based on extending the specification of these models with the possibility to define and use data ports. For the particular case of CCM, data ports are defined in an extended IDL3. We realized a prototype implementation based on classical CCM concepts, where the extended IDL3 is translatable to classical IDL3 definitions. To manage the shared data, different mechanisms were tested, like NFS and JUXMEM. This prototype is a proof of the concepts presented in Section 2.3 and of the facilities offered to the user. More details about the realized prototype as well as an application example using data ports can be found in [9].

3 Master-Worker Paradigm

3.1 Presentation

In the MASTER-WORKER programming paradigm, several instances of a same code (workers) have to be executed simultaneously with different parameter values sent by a master code. This paradigm is widely used in distributed and embarrassingly parallel applications like parametric applications. The relevance of this paradigm promoted numerous research activities to propose dedicated software environments. Examples are SETI@Home [11], XtremWeb [12] and BOINC [13] for Global Computing systems or DIET [14], NetSolve [15], Ninf-G [16] and Nimrod/G [17] for Network Enabled Server environments. A common objective of these environments is to lower the time taken by programmers to design, implement and deploy MASTER-WORKER applications. They offer transparent management of load balancing and dependability issues to enable efficient execution on a given computing infrastructure.

However, most of the proposed environments only focus on the MASTER-WORKER paradigm. Even if some of them provide another paradigm, like task farming in Ninf-G, they remain very specialized environments. Therefore, they are not sufficient to deal with multi-paradigm applications like code coupling applications.

3.2 Limitations with Existing Component Models

In opposition to MASTER-WORKER environments, existing component models require the designer to manage worker components (their number and load balancing.) Consequently, the code complexity is increased. This complexity is further increased since a designer has also to implement an adequate request transport policy. Such a policy

can be very complex and can depend on the underlying execution infrastructure. For instance, the policy is probably not the same when using PC clusters with thousands of processors or grids with heterogeneous processors and failures. Thus, code re-use is also limited.

As a result, existing component models do not offer a high level of abstraction to design those parts of an application that follow the MASTER-WORKER paradigm.

3.3 Collection Overview

In [18], we proposed to improve the support of the MASTER-WORKER paradigm in component models. For that, we proposed a high level MASTER-WORKER design model for which an overview is given in Figure 3. The proposal is based on the concept of *collection*. A collection is defined as a set of *exposed* ports, bound to some internal component type ports. A collection behaves like a component: it can be connected to other components and/or collections. However, such a composition is done in an *abstract* architecture description, which represents the user's view of the application. In this view, the number of component instances inside the collection as well as the mechanism to be used to distribute incoming requests are unknown. Ideally at deployment time – when resources are known – a collection is turned into a concrete assembly. Hence, at runtime, the collection is made of some internal component instances and of an instance of a *request transport pattern*. A *pattern* represents an implementation of an algorithm that specifies how to transfer requests from the *master* to *worker* components and how to schedule them. Its implementation is expected to be realized by experts and it may be based on software components. Figure 3 shows an example of a component based pattern (Round-Robin pattern) and a non component based one which reuses the already existing environment DIET. It also shows a collection instantiation with n workers and a Round-Robin pattern instance.

The interest of patterns is to enable the separation of request transport concerns. Several request transport algorithms, like Round-Robin, load balancing, request sequencing or others can be used within an application. Moreover, a pattern can be replaced by another without any other change in the application. The need of pattern replacement

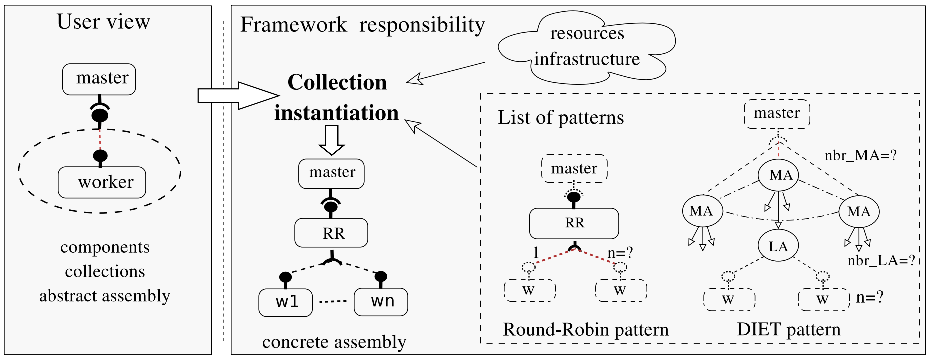


Fig. 3. Overview of the generic MASTER-WORKER model

appears when the number of requests or their loads lead to slow down the execution. In such a situation, the request transport algorithm and/or the number of workers may be a bottleneck. In [19] we studied how an adaptability framework can be integrated within the concept of collection to support dynamic modifications of collection's content. However, it is out of the scope of this paper.

3.4 Implementation

To illustrate the feasibility of the proposed generic model, we extended three specific component models: CCM [18], GCM/FRACTAL [18] and CCA [10]. Theses extensions provide specifications for collection and pattern concepts (description and usage). This section gives an overview of the CCM extension.

To describe a collection, the IDL3 of CCM was extended and a *Collection Description Language* (CDL) was defined. The extended IDL3 introduces a keyword `collection` to define a collection ports in a similar way as for components. The CDL allows the description of a collection content. To use a collection in an assembly, the CCM assembly language was extended with 14 new *XML* elements. This extension preserves the CCM composition principle.

To describe a pattern, many solutions are possible. We used for instance the *XSLT* language to both describe a pattern architecture and realize its introduction in a collection. As cited in Section 3.3, a pattern may be based on non component technology. In this case, its architecture introduces the concept of *adapter* components. An adapter component is a proxy component responsible to translate a master (resp. a MASTER-WORKER environment specific) request to the used MASTER-WORKER environment specific (resp. worker) request. Thus, the architecture of a pattern as well as a concrete assembly may be heterogeneous. To support such an assembly, we proposed an extension of the CCM assembly language. Such an assembly was realized for reusing DIET.

A set of experiments using the CCM extension was done to illustrate the benefits of the proposal. Results can be found in [20].

4 SPMD Parallel Components

4.1 Presentation

The *Single Program Multiple Data* (SPMD) paradigm is a paradigm where multiple instances of a single program are run in parallel, each one in its own process. The number of instances to use is a parameter of the application and these instances can communicate thanks to a well founded communication model based on message passing and collective communications.

This paradigm is well suited for scientific simulations where a meshing of the space to simulate is done. Each process works on a different set of meshes and the interaction at the boundaries are handled by message exchange between processes.

In order to couple SPMD codes it must be possible for each code to call methods implemented by others. The efficient implementation of such $M \times N$ method calls requires that no single process is in charge of the communications between the codes but rather that each process participate in the communication.

4.2 Limitations with Existing Component Models

Component models that do not natively support parallel components have no technical limitation that prevent component implementation to dynamically create new processes to implement parallel component. However, there is also no support for that. This means that the code handling the placement of new processes has to be replicated in every component. This also makes it difficult to obtain an optimal planning as each component instance does not have the informations about the placement of the processes of other component instances. Finally, this leads to the centralization of all interaction with a given component instance on the single process known by the component model which creates a bottleneck.

4.3 Parallel Component Overview

In [21], we proposed a model to allow the implementation and efficient coupling of parallel components. The implementation of a parallel component is done as a classical component implementation. The instantiation of such a component does however lead to the multiple instantiations of this implementation. The number of instances to create is a parameter set in the assembly.

Efficient $M \times N$ method calls are allowed by letting each process of the caller (resp. callee) provide (resp. receive) a part of each parameter and then receive (resp. provide) a part of the result. The ports used to couple the component are described as classical use/provide ports in what is called the user component interface description. The distribution of the data inside each component is a detail of the implementation and is therefore described in a side XML description that is part of the component implementation. The fact that ports of parallel component are described as classical use/provide ports also allows to handle the case where a non parallel component is connected to the port of a parallel component.

The implementation of the component is based on a component interface description that is slightly different from the user component interface: the internal component interface. In this interface, the parameters that have been described as distributed in the distribution description are replaced by their distributed equivalent data type (a matrix can for example be replaced by a vector for a (block(1),$*$) distribution).

4.4 Implementation

This model has been implemented as a CCM extension. The internal component interface is automatically generated together with a GridCCM glue layer and a manager by using the user component interface description and the XML data distribution description. The GridCCM glue code is then inserted in the user implemented component when it is compiled.

When the parallel component is instantiated, it is in fact the manager that is instantiated. This manager is then responsible for the multiple instantiations of the user implementation according to the parameters set in the assembly. Similarly, when a client gets a handle on a parallel component, it is in fact a handle on the manager. When two parallel components are connected, their manager transparently exchange information

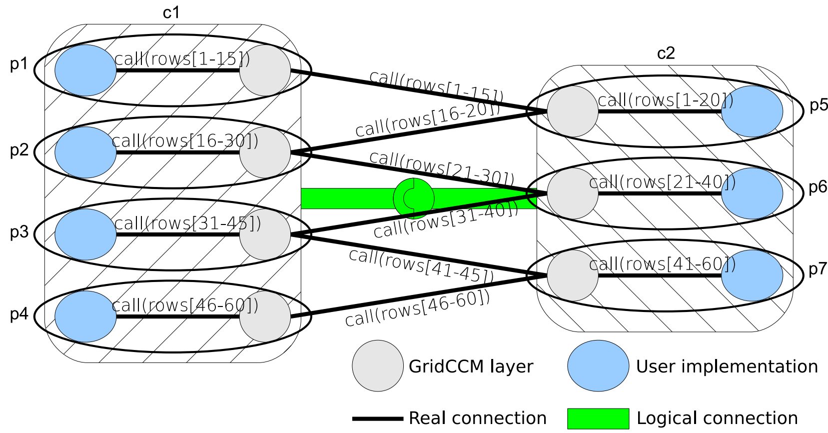


Fig. 4. Behavior of a method call in GridCCM

in order to connect the GridCCM layers of both components according to the redistribution that must be done.

When the user implementation calls a method on one of its use ports, the behavior is the one described in Figure 4. The call is intercepted by the GridCCM layer of the caller component that sends the data to the GridCCM layers of the callee component according to the data redistribution schema. Then this callee GridCCM layer calls the method on the callee user implementation. The same process is repeated for the return of the result of the method call.

5 Collective Communications

5.1 Presentation

As it has been said in the previous section, the most common interaction paradigm between processes of a SPMD parallel code is the use of message passing. Two kinds of operations exist: point to point operations that involve two processes (a sender and a receiver) and collective communication operations that involve a group of processes. As there is no assumption made on the number of processes with this latter kind of operations, it is well suited to SPMD codes where the number of processes is not known when the code is written.

The simplest collective operation is the *barrier* synchronization operation where each process is blocked in the call until all processes have called it. Other collective operations involve an exchange of data. For example in the *allgather* operation, each process receives a copy of the data of all processes in the group as shown in Figure 5 and in the *broadcast* operation, a specific process designed as the root sends its data to all the other processes in the group as shown in Figure 6.

Efficient implementation of these operations is greatly dependant on the resources on which the processes are executed and is still an active research domain. Effective implementation on distributed resources such as clusters requires communications to be done

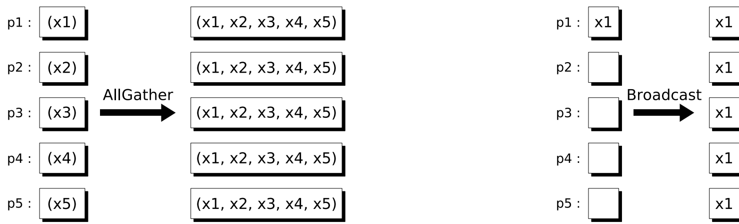


Fig. 5. The *allgather* collective operation

Fig. 6. The *broadcast* collective operation

in parallel as much as possible to take part of all communication links. There should therefore be no central process involved in all communications. In the case of grids, the different properties (namely bandwidth and latency) between intercluster links and the backbone link interconnecting clusters must also be taken into account.

5.2 Limitations with Existing Component Models

The next step after the introduction of SPMD parallel components as described in the previous section is to let the replicated entities in parallel components be implemented with components. In this case, communications between these instances has to be described by ports. While it is quite easy to reuse existing communication paradigms such as events or remote method call to provide a semantic similar to point to point message passing, there is no straightforward solution for the use of collective communication operations.

5.3 Collective Communications Model

In [22], we proposed a model to use collective communication operations between components. A component that use collective communication has to describe a use port with a dedicated `CollComm` interface. This interface is very similar to the MPI interface in order to ease the transition from MPI to component collective communication except for the groups that are not described by a parameter for each call. Instead, groups are described in the assembly, they are created by the instantiation of a `CollCommProvider` component that provides the `CollComm` interface. The component instances whose use ports are connected to the provide port of this instance are part of the group. A component instance can be part of more than one group if it declares several use ports of the `CollComm` interface as shown in Figure 7.

5.4 Implementation

Collective communications have been implemented as a CCM extension. As a classical centralized implementation of the `CollCommProvider` component would lead to bottlenecks, the component implementation model has been extended to allow efficient implementations. Some concepts have been added: *AnyToAny* connections and *replicating* implementations.

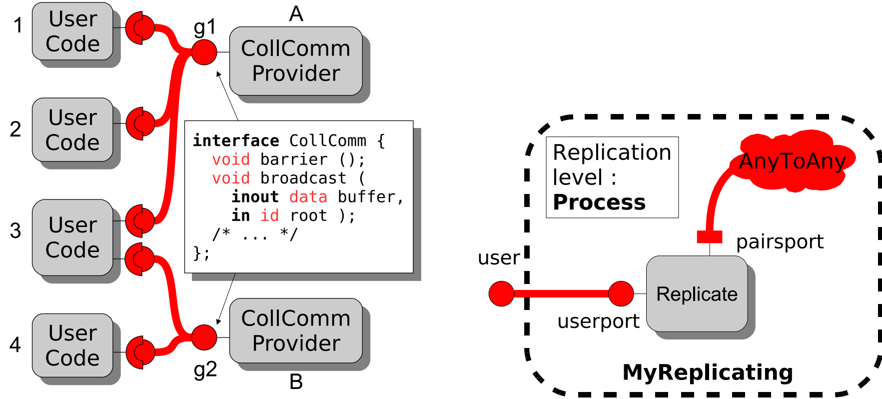


Fig. 7. Collective communication usage in the assembly

Fig. 8. A replicating component

A *AnyToAny* connection provides a semantic similar to use/provide connections but with any number of component participating. Each component has to provide the interface of the connection, in return it can call an operation of the interface on any component participating in the connection.

A *replicating* component implementation is defined by a *replicate* component, a *replication level* (process, node, cluster, grid ...), internal *AnyToAny connections* and a *mapping* of the ports of the component to ports of the replicate as shown in Figure 8. At runtime, the component is replaced by a set of instances of the replicate. The number of instances is determined by the replication level and the usage of the component. For example if the replication level is set to process and that components connected to a provides port of the component are spread amongst three different processes, then three instances of the replicate will be created, one on each process where the component is used. Internal *AnyToAny* connections are used to connect all the replicate instances and the connections to the ports of the component are replaced by connection to the mapped ports.

6 Conclusion

Software component appears to be a very promising concept to be a *programming* entity. It allows to improve productivity, to reduce complexity and to hide implementation details. This paper showed how four important abstractions for scientific applications can fit well into component models without impacting performance. These four abstractions impacted the various levels of a component model: data sharing can be provided through a new kind of port while parallel components require a new kind of component; the master-worker paradigm and the collective communications mainly involve the assembly level.

However, this paper does not claim to be exhaustive. In particular, this paper does not deal with abstractions that required to modify the kind of the assembly level such as workflow [23] or skeletons [24].

Future researches can be divided into two branches. First, most of the abstractions requires parameter selection to perfectly fit to the actual resources. Hence, strategies and algorithms are needed to achieve automatically selection. This is particular very important for volatile resources so as to have self-$\star$ implementations of the abstractions. Second, the implementation of all these abstractions into an efficient and coherent runtime is challenging. Moreover, new abstractions may appear. Thus, we need a flexible mechanism. A promising technique is to use model transformation to transform programming abstractions to execution entities.

References

1. Szyperski, C., Gruntz, D., Murer, S.: Component Software - Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley/ACM Press (2002)
2. OMG: CORBA component model, v4.0. Document formal/2006-04-01 (April 2006)
3. Beisiegel, M., Blohm, H., Booz, D., Edwards, M., Hurley, O., Ielceanu, S., Miller, A., Karmarkar, A., Malhotra, A., Marino, J., Nally, M., Newcomer, E., Patil, S., Pavlik, G., Raepple, M., Rowley, M., Tam, K., Vorthmann, S., Walker, P., Waterman, L.: SCA Service Component Architecture - Assembly Model Specification, version 1.0. Technical report, Open Service Oriented Architecture collaboration (OSOA) (March 2007)
4. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The fractal component model and its support in java. Software Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems 36(11-12) (2006)
5. Institute, P.M.: Basic features of the grid component model. CoreGRID Delivrable D.PM.04, CoreGRID (March 2007)
6. Bernholdt, D.E., Allan, B.A., Armstrong, R., Bertrand, F., Chiu, K., Dahlgren, T.L., Damevski, K., Elwasif, W.R., Epperly, T.G.W., Govindaraju, M., Katz, D.S., Kohl, J.A., Krishnan, M., Kumfert, G., Larson, J.W., Lefantzi, S., Lewis, M.J., Malony, A.D., McInnes, L.C., Nieplocha, J., Norris, B., Parker, S.G., Ray, J., Shende, S., Windus, T.L., Zhou, S.: A component architecture for high-performance scientific computing. International Journal of High Performance Computing Applications 20(2), 163–202 (2006)
7. Aldinucci, M., Campa, S., Danelutto, M., Vanneschi, M., Dazzi, P., Laforenza, D., Tonellotto, N., Kilpatrick, P.: Behavioural Skeletons in GCM: Autonomic Management of Grid Components. In: Baz, D.E., Bourgeois, J., Spies, F. (eds.) Proc. of Intl. Euromicro PDP 2008: Parallel Distributed and Network-based Processing, Toulouse, France, pp. 54–63. IEEE, Los Alamitos (2008)
8. Antoniu, G., Bougé, L., Jan, M.: JuxMem: An adaptive supportive platform for data sharing on the grid. Scalable Computing: Practice and Experience 6(3), 45–55 (2005)
9. Antoniu, G., Bouziane, H., Breuil, L., Jan, M., Pérez, C.: Enabling transparent data sharing in component models. In: 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID), Singapore, May 2006, pp. 430–433 (2006)
10. Antoniu, G., Bouziane, H.L., Jan, M., Pérez, C., Priol, T.: Combining data sharing with the master-worker paradigm in the common component architecture. In: The 15th IEEE International Symposium on High Performance Distributed Computing (HPDC), Paris, France (June 2006)
11. Anderson, D., Bowyer, S., Cobb, J., Gebye, D., Sullivan, W., Werthimer, D.: A new major SETI project based on Project SERENDIP data and 100,000 personal computers. In: Conference Paper, Astronomical and Biochemical Origins and the Search for Life in the Universe, IAU Colloquium 161, p. 729. Bologna, Italy (1997)

12. Germain, C., Néri, V., Fedak, G., Cappello, F.: XtremWeb: building an experimental platform for Global Computing. In: Buyya, R., Baker, M. (eds.) GRID 2000. LNCS, vol. 1971, pp. 91–101. Springer, Heidelberg (2000)
13. Anderson, D.P.: Berkeley Open Infrastructure for Network Computing (2002), `http://boinc.berkeley.edu/`
14. Caron, E., Desprez, F., Lombard, F., Nicod, J., Quinson, M., Suter, F.: A Scalable Approach to Network Enabled Servers. In: Monien, B., Feldmann, R.L. (eds.) Euro-Par 2002. LNCS, vol. 2400, pp. 907–910. Springer, Heidelberg (2002)
15. Casanova, H., Dongarra, J.: NetSolve: A Network-Enabled Server for Solving Computational Science Problems. The International Journal of Supercomputer Applications and High Performance Computing 11(3), 212–223 (1997)
16. Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T., Matsuoka, S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. J. Grid Compututing 1(1), 41–51 (2003)
17. Buyya, R., Abramson, D., Giddy, J.: Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. High-Performance Computing 01(1), 283 (2000)
18. Bouziane, H.L., Pérez, C., Priol, T.: Modeling and executing master-worker applications in component models. In: 11th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS), Rhodes Island, Greece (April 2006)
19. André, F., Bouziane, H.L., Buisson, J., Pazat, J.L., Pérez, C.: Towards dynamic adaptability support for the master-worker paradigm in component based applications. TR RT-0333, INRIA (April 2007)
20. Bouziane, H.: De lábstraction des modèles de composants logiciels pour la programmation dápplications scientifiques distribuées. Ph.D thesis, Université de Rennes 1, IRISA/INRIA, Rennes, France (February 2008)
21. Pérez, C., Priol, T., Ribes, A.: A parallel corba component model for numerical code coupling. In: Parashar, M. (ed.) Proc. 3rd International Workshop on Grid Computing. LNCS, vol. 17, pp. 88–99. Springer, Heidelberg (2000); Special issue Best Applications Papers from the 3rd Intl. Workshop on Grid Computing
22. Bigot, J., Pérez, C.: Enabling collective communications between components. In: CompFrame 2007: Proceedings of the 2007 symposium on Component and framework technology in high-performance and scientific computing, pp. 121–130. ACM Press, New York (2007)
23. Bouziane, H., Pérez, C., Priol, T.: A software component model with spatial and temporal compositions for grid infrastructures. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 698–708. Springer, Heidelberg (2008)
24. Aldinucci, M., Bouziane, H., Danelutto, M., Pérez, C.: Towards software component assembly language enhanced with workflows and skeletons. In: Joint Workshop on Component-Based High Performance Computing and Component-Based Software Engineering and Software Architecture (CBHPC/COMPARCH 2008), October 14-17 (2008)

Group Abstractions for Organizing Dynamic Distributed Systems

José C. Cunha, Carmen P. Morgado, and Jorge F. Custódio

CITI, Dep. Informática – FCT, Universidade Nova de Lisboa

Abstract. We discuss the usefulness of group-based abstractions for the modeling of distributed applications. We suggest that groups can be considered at a higher level of abstraction, and group concepts can be supported as distributed programming constructs in high-level models and languages. We illustrate this approach by discussing GroupLog, an abstract model that uses groups to specify a structured space of interactions, and the collective behavior of cooperating group members.

1 Introduction

Distributed system applications and systems, as illustrated by the Web, P2P, Mobile, and the Grid, now exhibit more tightly-coupled interactions, new forms of dynamic behavior, and varying scale in terms of their components. Group models allow their structuring in terms of well-identified collections of autonomous entities. Groups help to manage the logical characteristics that are shared by the group members, such as common computational or communication behaviors, common goals in multi-agent systems, access to common resources and information, cooperation to ensure service functionalities under specific performance, quality of service, or cost constraints. Group models also offer potentially scalable solutions, by allowing hierarchies, and exploiting global and local coordination and communication strategies.

Group based communication has been a topic of intense investigation in the past decades, addressing issues such as efficient multicast protocols in LANs and WANs, atomicity and causality in the semantics of message delivery, consistency of distributed process views, transparent fault-tolerance and replication, and management of dynamic process groups [16,13,9]. As a result, significant abstractions have been proposed and consolidated in distributed group programming libraries and platforms. As a significant example, among others, we note the influence of abstractions such as virtual synchrony [8,2,17], and their integration into programming libraries and toolkits [3]. More recently, there have been efforts to support flexible adaptation of group concepts to different applications and environments, for instance exercising stronger or weaker forms of consistency, different semantics for message delivery, efficient handling of large-scale groups, or ad-hoc groups in mobile environments [10,23,9,17,21].

Such long-term research had a significant influence on a diversity of application domains requiring online collaboration between distributed entities

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 450–459, 2009.
© Springer-Verlag Berlin Heidelberg 2009

(users or software agents). Examples can be drawn from multiple fields, namely for computer supported cooperative work (CSCW), groupware, multi-agent systems, and peer-to-peer systems. A more recent trend, related to online interaction and information sharing in social networks, is increasing the pressure to improve the computational support for group abstractions.

In summary, groups provide an adequate approach to handle scalability, mobility and dynamism in a distributed system. Although there have been several attempts to exploit the above dimensions at different contexts and with distinct levels of concern, many aspects still need improvement towards a more effective use of groups as an organisation and cooperation paradigm. In particular, we claim there is the need and opportunity to raise the level of abstractions provided by group-based models, in order to address the high-level specification of applications or the organization of services in a large-scale distributed computing system.

In this paper, we discuss, from a global perspective, the importance of abstractions and models (section 2), followed by a review of the relevance of group-based abstractions as an approach to application distribution and organization (section 3). We present a brief outline of the GroupLog approach, its implementation and applications. The paper concludes by discussing open issues and future research (section 5).

2 Abstraction and Models

The importance of adequate abstractions is well recognized, as well as the difficulty to achieve an appropriate compromise between the transparency level provided by each abstraction and its efficient implementation under current technology constraints. Concerning distributed computing abstractions, there is now a significant body of abstractions that address several critical dimensions, for example related to time, communication, failure, composition of services, decomposition and distribution of computation and data, coordination of distributed processes, and more recently service and resource abstractions. Also, for some application domains, specific abstractions were incorporated into support problem-solving environments.

Emerging application and computing enviornments have raised new challenges and open issues. For example, how to design large-scale applications and systems with hundreds or thousands of distributed entities, concerning their dynamic organizations and problem-solving approaches. Also, how to support novel forms of collaboration, with a multiplicity of observers and controllers, for example in virtual organizations or social networks, with highly dynamic behaviors.

An important concern is to try to understand how such issues are better supported by high-level programming abstractions and models, and how these models can be mapped into intermediate frameworks and lower level programming interfaces.

3 Group-Based Abstractions

We briefly review the main dimensions of group-based abstractions and their relevance to support the structuring of distributed applications. See [9,13] for compreehensive surveys.

3.1 Group Abstractions

A group is a collection of cooperating entities, addressed by a unique and global name, as a single entity in the system. In a general model, a group can have a hierarchical structure, including elementary entitities and composite entities (that is, groups), as members. In dynamic groups, members can enter and leave during the group lifetime. Group models may provide support for the group members to observe common and consistent views of the group history. The group history is defined by an agreed ordering on the set of events related to modifications of the group membership, and to the communication events within the group and with its environment. Depending on the applications, there are different requirements for the consistency degrees for the group views, ranging from the strong guarantees provided by virtual synchrony models, to their absence. Different application scenarios, eg with mobility or large-scale, require distinct semantics for the views consistency, as well as the ordering of message delivery to the group members. Several platforms and programming APIs [3,1,23] support a flexible composition of the group communication protocols, allowing the application to select the appropriate semantics.

As mentioned, dynamic process groups have been the subject of a long-term research and a compreheensive survey is out of the scope of this paper. Group models have been exploited to support applications and services in distributed systems, by relying on the cooperation among group members to ensure improved reliability, fault-tolerance and performance through replication, parallelism or decomposition of work, internal to the group [17,9].

We believe there are still aspects that remain to be explored regarding the use of groups as an organisation paradigm at a higher level of abstraction, to exploit scale, dynamism, shared knowledge, and information, and cooperation.

3.2 Groups as Distributed Programming Abstractions

The group concept has been intensively explored at the operating system and the middleware levels [9], and has been used for application development, but comparatively fewer proposals have attempted its integration into high-level programming frameworks [14,18,22,4,6,5,11].

Groups can be considered at a higher level of abstraction, and group concepts can be supported as distributed programming constructs in high-level models and languages. Group abstractions can be defined neutral to any specific programming platform or system architecture, in order to allow their adaptation to a diversity of contexts. Multiple instantiations of such high-level abstractions

can then be defined, and their semantics adapted according to the programming language framework, to the system architecture characteristics, and to the application scenarios.

In a general group model, clients may address the group as an atomic entity in a transparent way, by addressing a well-defined group interface corresponding to a set of entry-points. The group's internal behavior can be hidden from the outside. For example, the group can support a reactive behavior, on reaction to external invocations of its interface, or it can have a pro-active, goal-oriented behavior. The clear separation between the group interface and the internal behavior allows to exploit local policies, internal to a group, in a transparent way. The internal organization of the group in terms of multiple member entities allows to exploit cooperation, shared state, or to manage components with common properties.

Group specification, identification and discovery. In order to develop a flexible framework to exploit the diversity of the dimensions associated to group models, we consider two levels of concern.

Group specification. At this level, abstractions are defined for the organization of distributed systems and applications in terms of groups of entities. In order to allow distinct instantiations, each elementary entity can have different semantic interpretations, depending on the considered active computational entities, such as processes, objects, agents, or services. The model allows the specification of a group as a structuring unit, with a public interface, that can be exposed as a set of methods, for example in a object-oriented context, or as a set of ports, in a service-oriented context. The model also defines primitives for dynamic group management, communication among the group members, and a semantics for the consistency of views observed by the group members. Different forms of communication must be supported, in order to capture the most commons patterns of interaction occurring in applications. In particular, the group members must have transparent access to a form of group shared state.

Group identification and discovery. At this level, strategies are defined for the dynamic identification and discovery of groups in a distributed environment.

In a first step, we identify the relevant attributes that guide dynamic group identification and formation.

As well-identified patterns of behavior emerge in a distributed environment, or common goals are dynamically identified by multiple cooperating agents, groups can be automatically generated by a system, in response to the detection of relevant events. The motivation behind this perspective is to provide a mechanism to help capture and identify common attributes in distributed and dynamically evolving entities. The identification can be guided by the definition of the attributes that represent the common profiles of the members of each type of group. The relevant groups can be defined by an initial list of distinctive attributes, which are then used by a group discovery mechanism to detect potential candidate members, and to automatically aggregate them into the identified groups.

In this second step, we consider global coordination strategies, by creating or eliminating groups and dynamically managing their membership. Groups may be created or destroyed, and their lifetime managed, depending on the information kept and updated in information repositories. It is possible to consider short-term policies, based on information on local repositories, and more global long-term policies, based on a higher-level interpretation of aggregated information.

This can be useful to dynamically form ad-hoc groups which may be due to a spontaneous definition of communities of interests, like the ones formed by geographical proximity of mobile users. It also enables to exploit a dynamic management of the cooperation among distributed entities, in reaching common goals, or in sharing common knowledge and functionalities. This approach is exploited in the MAGO system (Section 4) to identify common user interests and profiles in a dynamic environment supporting collaborative mobile users.

The above strategies can also be useful to guide strategies for autonomic management of complex distributed systems and applications. For example, to support intelligent strategies for optimization of resource management depending on cost and resource usage characteristics. In related work, we have exploited this idea for the coordination of utility managed multi-agent groups [7]. In this system, in a first stage multiple selfish agents each try to optimize their local utilities. Then, in a second stage, groups are dynamically formed by agents trying to identify partners which enable the agents to achieve a higher local utility through collaboration among group peers.

Similar concerns appear in related works (see [21] for a survey of ongoing research). For example, Abramson and Mittu [21] , have identified a number of issues for managing agent groups in large scale distributed open environments: dynamic group formation, role allocation, synchronisation of beliefs, communication selectivity, information sharing.

4 The GroupLog Approach

As an illustration of the above mentioned approach, a brief presentation of GroupLog is given in the following. Then three experiments are described, as an application of the approach in distinct contexts.

4.1 Overview

The GroupLog approach is based on the following main principles:

- Two main program structuring entities are defined: elementary entities, and groups, as collective entities. These concepts capture respectively, the computation and the collective coordination aspects, and are defined in an orthogonal way.
- An elementary entity represents the basic computational aspect that is encapsulated within an autonomous program unit. It has a well-defined public interface that hides its internal behavior. Different interpretations are allowed for an elementary entity depending on the programming framework,

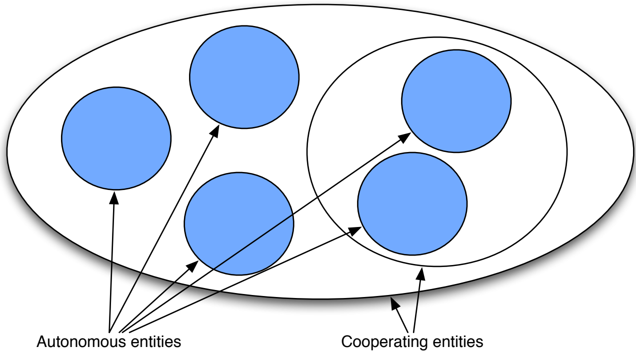


Fig. 1. GroupLog entities

as a process, an object, an agent, or a service, its interface entry points being interpreted as a set of methods, predicates or ports.

– An entity exhibits a well-defined behavior, depending on its current state and on the invoked interface entry point. The GroupLog approach is neutral concerning the semantics defining the internal behavior of each elementary entity. This definition is left to each specific instantiation of the model that must specify the behavior concerning the creation and elimination of elementary entities, as well as the actions in response to the invocation of the interface. This allows interpretations of elementary entities as reactive elements, or as agents with pro-active behavior and internal concurrency.
– Groups are considered as programming units supporting an organization and cooperation paradigm, and may be structured forming hierarchies. A group has a well-defined public interface for interaction with its environment, that is defined in the same way as the interface of an elementary entity. Thus groups and elementary entities are not distinguished from their outside. Separation between the group interface and its internal behavior allows implementing local policies within a group, in a transparent way.
– Both elementary and collective entities can have multiple instances that can be created and destroyed dynamically. Group membership changes dynamically, as new entities can enter and leave the groups. The group internal behavior is defined by the collective behaviors of its members. A group hides its membership to its environment but allows the internal redirection of communication through the group interface. An entity can belong to one or more groups, so it can inspect the list of its current groups and their membership.
– Groups encapsulate confined interaction spaces, and allow interactions among group members to be more easily managed due to smaller scales, thus enabling more appropriate coordination paradigms. In particular, interactions within the group rely on different forms of communication that can be combined: point-to-point, multicast, and access to a shared group space. The latter is represented as a multi–set of tuples and is accessed by primitives based on the Linda coordination model[12]. GroupLog integrates

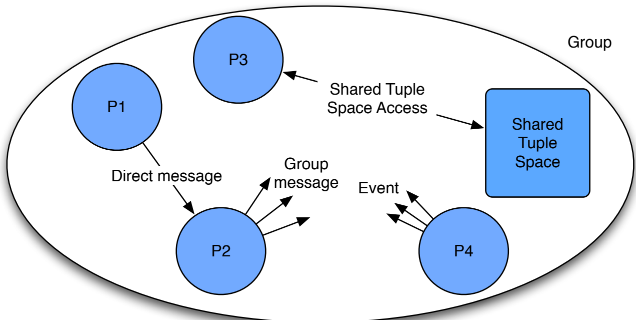


Fig. 2. JGroupSpace

the above types of interactions in the context of a dynamic group membership, in order to guarantee the consistency of the views that are observed by the group members, according to a virtual synchrony model [8].

4.2 Instances of the GroupLog Model

A GroupLog system is a collection of distributed entities, that communicate through well-defined interfaces, and join groups to participate in collective activities, and having access to shared spaces within each group.

Distinct instances of the model have been defined and implemented at distinct abstraction levels:

- A logic-based instance of GroupLog [4,6,5] exploits the expressiveness and declarativeness of logic programming. It finds applications in areas where there is the need of an inference capability modeled by logic based agents, and a multi-agent coordination model. GroupLog in logic is defined at two levels: L_1, defines an agent as a logic entity with well-defined interface, knowledge and behavior, and L_2 defines groups of agents. The implementation relies on a PVM-Prolog layer that we have implemented for group management, communication and shared space management.
- An object-oriented instance of GroupLog, JgroupSpace [15], defines a Java API for group management, message and event based communication, and access to a group shared tuple space. It relies on a distributed implementation on top of JGroups[3,1].
- A high-level model for collaborative interactive applications, MAGO [19,20], exploits the GroupLog approach to support the modeling of distributed applications where multiple entities exhibit different forms of interaction and mobility. Collaboration and sharing of information are supported in the context of dynamically formed groups as illustrated in Figure 3. The MAGO system supports the concept of implicit groups, for the automatic identification and creation of groups, related to the dynamic detection of users'

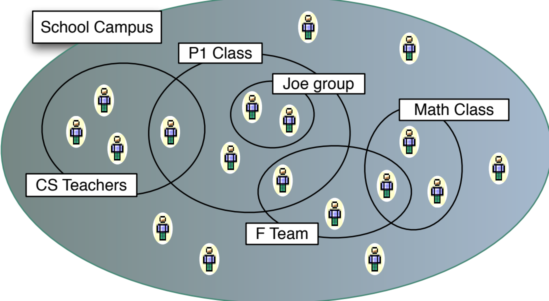


Fig. 3. MAGO example

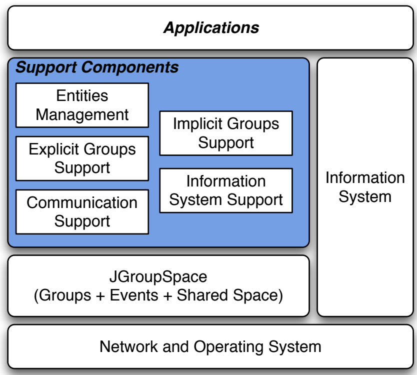


Fig. 4. Layers presented in MAGO

common profiles and interests. MAGO is implemented as a layered architecture (Figure 4), with an interface to an external information system, and relies on JGroupSpace as the supporting platform.

5 Conclusions and Future Work

Emerging large-scale and dynamic distributed applications and computing environments pose a diversity of challenges. Namely they require hierarchical and dynamic organizations, confinement of local and global policies, more flexible and efficient forms of communication and information sharing, varying scale and locality (physical and logical), distinct semantics for interaction with distinct consistency guarantee, and knowledge-based approaches.

From our experiments with GroupLog, we conclude it is desirable to define group concepts at a high-level of abstraction, and then to map and adapt them to specific programming language frameworks and application scenarios. In order to

provide a flexible framework to ease such mappings, we have identified two levels of concern, one related to the specification of the group models, and the other related to the mechanisms and strategies for group identication and management. Such framework should allow distinct semantics for the group concepts, in order to enable, for example, stronger or weaker semantics for the consistency of the group views. While several group communication platforms already provide significant support for example concerning the composition of distinct group protocols, there is still the need to provide increased flexibility to adapt to distinct application scenarios.

In our future work, we consider the evolution of GroupLog abstractions to model large-scale distributed Grid applications, by exploiting decentralized coordination strategies and concepts emerging from social networks, to increase the efficiency of coordinating large groups, and to provide high-level semantic models for group behavior [21].

Acknowledgments

To Fernanda Barbosa for her work in GroupLog in logic, to Omer Rana and Stephen Lynden for joint work with utility-based groups. The work on GroupLog was partially supported by CITI, funded by FCT/MCTES. Thanks to Shantenu Jha for the encouragement to proceed with this work.

References

1. JGroups tutorial, `http://www.jgroups.org/`
2. Babaoglu, I.: On programming with view synchrony. In: ICDCS 1996: Procs of 16th International Conference on Distributed Computing Systems, p. 3. IEEE Computer Society, Los Alamitos (1996)
3. Ban, B.: Design and implementation of a reliable group communication toolkit for Java. Technical report, Cornell University (1998)
4. Barbosa, F.: Distributed programming abstractions based on groups: the GroupLog model (in portuguese). Ph.D thesis, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, Lisboa (2003)
5. Barbosa, F., Cunha, J.C.: A language framework for group based multi-agent systems: GroupLog. In: Proceedings of CABS, Workshop on the Foundations and Applications of Collective Agent Based Systems, ESSLLI-99 11th European Summer School in Logic, Language and Information. Utrecht University (1999)
6. Barbosa, F., Cunha, J.C.: A coordination language for collective agent-based systems: GroupLog. Applied Artificial Intelligence 15(1), 59–78 (2001)
7. Barbosa, F., Cunha, J.C., Rana, O., Lynden, S.: Coordination in Utility Managed Multi-Agent Group. In: Yang, L.T., Pan, Y. (eds.) Advances in Computation: Theory and Practice, vol. 15, pp. 209–219. Nova Science (2004)
8. Birman, K., Joseph, T.: Exploiting virtual synchrony in distributed systems. In: Procs. of the 11th ACM Symposium on Operating systems principles, pp. 123–138. ACM Press, New York (1987)
9. Birman, K.P.: Reliable Distributed Systems Technologies, Web Services, and Applications. Springer, Heidelberg (2005)

10. Birman, K.P., Friedman, R., Hayden, M., Rhee, I.: Middleware support for distributed multimedia and collaborative computing. Softw. Pract. Exper. 29(14), 1285–1312 (1999)
11. Caromel, D., Henrio, L., Cardelli, L.: Theory of Distributed Objects. Springer, Heidelberg
12. Carriero, N., Gelernter, D.: Linda in context. Commun. ACM 32(4), 444–458 (1989)
13. Chockler, G.V., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. ACM Comput. Surv. 33(4), 427–469 (2001)
14. Cruz, J.-C., Ducasse, S.: A group based approach for coordinating active objects. In: Ciancarini, P., Wolf, A.L. (eds.) COORDINATION 1999. LNCS, vol. 1594, pp. 355–370. Springer, Heidelberg (1999)
15. Custódio, J.: Jgroupspace - support for distributed group-based programming (in portuguese). M.Sc. thesis, Universidade Nova de Lisboa, Faculdade de Ciëncias e Tecnologia, Lisboa (2008)
16. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. 36(4), 372–421 (2004)
17. Van Renesse, R., Birman, K.P.: Reliable Distributed Computing with the Isis Toolkit. IEEE Computer Society Press, Los Alamitos (1994)
18. Maffeis, S.: The object group design pattern. In: COOTS 1996: Procs. of 2nd USENIX Conf. on Object-Oriented Technologies, p. 12. USENIX Association (1996)
19. Morgado, C., Correia, N., Cunha, J.C.: A group-based approach for modeling interactive mobile applications. In: International ACM Conference on Supporting Group Work - Group 2007 - Proceedings. ACM Press, New York (2007)
20. Morgado, C.P.: A group model for distributed interactive applications (in portuguese). Ph.D thesis, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, Lisboa (2008)
21. Mailler, R., Scerri, P., Vincent, R.: Coordination of large-scale multi-agent systems. Springer, Heidelberg (2006)
22. Pardyak, P., Bershad, B.N.: A group structuring mechanism for a distributed object-oriented language. In: International Conference on Distributed Computing Systems, pp. 312–319 (1994)
23. van Renesse, R., Birman, K., Hayden, M., Vaysburd, A., Karr, D.: Building adaptive systems using Ensemble. Softw. Pract. Exper. 28(9), 963–979 (1998)

Author Index

Abbes, Heithem 235
Aldinucci, M. 403
Alexander, Michael 1
Anthes, Christoph 327, 352, 382
Asenjo, R. 273
Atoofian, Ehsan 152
Augonnet, Cédric 174

Baliś, Bartosz 329
Baniasadi, Amirali 152
Benkner, Siegfried 141
Bertogna, Mario Leandro 13
Bigot, Julien 438
Bilardi, Gianfranco 127
Bjerke, Håvard K.F. 3
Bouziane, Hinde Lilia 438
Bressler, Helmut 352
Bubak, Marian 329

Caillat, Gabriel 247
Caniou, Y. 223
Caromel, Denis 423
Castillo, R. 273
Cérin, Christophe 195
Childs, Stephen 1
Coady, Yvonne 152
Cole, Murray 401
Corbera, F. 273
Cunha, José C. 450
Custódio, Jorge F. 450

Danelutto, M. 403
De Giusti, Armando 13
Diakhaté, François 53
Donaldson, Alastair F. 163
Dubacq, Jean-Christophe 235
Duc, Guillaume 211

Engelmann, Christian 63

Fahringer, Thomas 327, 361
Fedak, Gilles 247
Ferris, J. 371
Fontán, Javier 23
Forsell, Martti 123

Funika, Włodzimierz 329
Fürlinger, Karl 263

Gallard, Jérôme 43
Gallard, Pascal 43
Gay, J.-S. 223
Gholami, Ali 113
Glinka, Frank 371, 390
Gorlatch, Sergei 390
Grosclaude, Eduardo 13
Grünter, E. 94
Gudenkauf, S. 103
Guim, Francesc 199
Guzy, Krzysztof 329

Habib, Irfan 3
Hasselbring, W. 103
He, Haiwu 247
Hedman, Fredrik 113
Herbert, Jordan 361
Höing, A. 103
Holub, Petr 339
Huck, Kevin A. 283
Huedo, Eduardo 23

Iwainsky, Christian 315

Jesshope, Chris 129
Jha, Shantenu 401
Jourdren, Herve 53
Juurlink, Ben 184

Kao, O. 103
Katz, Dan 401
Keir, Paul 163
Keller, Jörg 131
Kertész, Attila 199
Keryell, Ronan 211
Kessler, Christoph W. 131
Kilpatrick, P. 403
Kluge, Michael 295
Knüpfer, Andreas 261, 295
Kranzlmüller, Dieter 327

Landertshamer, Roland 352, 382
Lèbre, Adrien 43

Leyton, Mario 423
Liška, Miloš 339
Llorente, Ignacio M. 23
Lodygensky, Oleg 247
Lokhmotov, Anton 163
Luque, Emilio 13

Malony, Allen D. 283
Meenderinck, Cor 184
Mehofer, Eduard 141
Meier, M. 94
Memon, A. Shiraz 113
Memon, M. Shahbaz 113
Mey, Dieter an 315
Mickler, Holger 295
Mix, Hartmut 84
Montero, Rubén S. 23
Moore, Shirley 263
Morgado, Carmen P. 450
Morin, Christine 43
Morris, Alan 283
Moskovsky, Alexander A. 33
Mucci, Phil 263
Mucci, Phillip 113
Müller, Matthias S. 261, 295

Nae, Vlad 361
Nagel, Wolfgang E. 295
Naiouf, Marcelo 13
Namyst, Raymond 53, 174
Nasser, Bassem I. 382
Natvig, Lasse 141
Naughton, Thomas 63
Navarro, A. 273
Netzer, Gilbert 113
Niederberger, R. 94

Oistrez, T. 94
Ong, Hong 63
Ostropytskyy, Vitaliy 84

Parashar, Manish 401
Perache, Marc 53
Pérez, Christian 438
Pervin, Artem Y. 33
Pllana, Sabri 141
Ploss, Alexander 390
Priol, Thierry 438
Prodan, Radu 361

Rana, Omer 401
Rasch, Katharina 84
Riedel, Morris 113
Roch, Jean-Louis 211
Rodero, Ivan 199
Rodríguez, Manuel 23
Romberg, Mathilde 84

Scherp, G. 103
Schöne, Robert 84
Schuller, Bernd 75
Schumacher, Miriam 75
Scott, Stephen L. 43, 63
Shah, Zeeshan A. 113
Shende, Sameer 283
Shiyachki, Dimitar 3
Spear, Wyatt 283
Streit, Achim 73
Surridge, Mike 371, 382
Szebenyi, Zoltán 305
Szepieniec, Tomasz 329

Talia, Domenico 415
Tapiador, Daniel 23
Terpstra, Dan 263
Tikotekar, Anand 63
Träff, Jesper Larsson 123

Unterkircher, Andreas 3
Urbah, Etienne 247

Vallée, Geoffroy 43, 63
Varrette, Sebastien 211
Volkert, Jens 352, 382

Walker, Bruce J. 33
Wismüller, Roland 329
Wolf, Felix 305
Wylie, Brian J.N. 305

Xhafa, Fatos 141

You, Haihang 263

Zapata, E.L. 273
Ziegler, Wolfgang 73