Hai-Xiang Lin Michael Alexander
Martti Forsell Andreas Knüpfer
Radu Prodan Leonel Sousa
Achim Streit (Eds.)

LNCS 6043

Euro-Par 2009 Parallel Processing Workshops

HPPC, HeteroPar, PROPER, ROIA, UNICORE, VHPC 2009
Delft, The Netherlands, August 2009
Workshops

Springer

Hai-Xiang Lin   Michael Alexander
Martti Forsell   Andreas Knüpfer
Radu Prodan   Leonel Sousa   Achim Streit (Eds.)

Euro-Par 2009 Parallel Processing Workshops

HPPC, HeteroPar, PROPER, ROIA, UNICORE, VHPC
Delft, The Netherlands, August 25-28, 2009
Workshops

Springer

Volume Editors

Hai-Xiang Lin
Delft University of Technology, 2628 Delft, The Netherlands
E-mail: h.x.lin@tudelft.nl

Michael Alexander
Scaledinfra technologies, 1010 Vienna, Austria
E-mail: maxlexand@scaledinfra.com

Martti Forsell
VTT, 90570 Oulu, Finland
E-mail: martti.forsell@vtt.fi

Andreas Knüpfer
Technische Universität Dresden, 01069 Dresden, Germany
E-mail: andreas.knuepfer@tu-dresden.de

Radu Prodan
Technical University of Innsbruck
6020 Innsbruck, Austria
E-mail: radu@dps.uibk.ac.at

Leonel Sousa
Instituto Superior Técnico/INESC-ID. 1000-029 Lisbon, Portugal
E-mail: leonel.sousa@gmail.com

Achim Streit
Jülich Supercomputing Centre, 52425 Jülich, Germany
E-mail: a.streit@fz-juelich.de

Library of Congress Control Number: 2010929253

CR Subject Classification (1998): C.4, D.2, C.2, D.4, C.2.4, C.3

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

ISSN	0302-9743
ISBN-10	3-642-14121-8 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-14121-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

Euro-Par is an annual series of international conferences dedicated to the promotion and advancement of all aspects of parallel and distributed computing. Euro-Par 2009 was the 15^{th} edition in this conference series. Througout the years, the Euro-Par conferences have always attracted high-quality submissions and have become one of the established conferences in the area of parallel and distributed processing. Built upon the success of the annual conferences and in order to accommodate the needs of special interest groups (among the conference participants), starting from 2006, a series of workshops in conjunction with the Euro-Par main conference have been organized. This was the fifth year in which workshops were organized within the Euro-Par conference format.

The workshops focus on advanced specialized topics in parallel and distributed computing. These topics reflect new scientific and technological developments. While the community for such new and specific developments is still small and the topics have yet to become mature, the Euro-Par conference offers a platform in the form of a workshop to exchange ideas and discuss cooperation opportunities.

The workshops in the past four years have been very successful. The number of workshop proposals and the number of finally accepted workshops have gradually increased since 2006. In 2008, nine workshops were organized in conjunction with the main Euro-Par conference. In 2009, there were again nine workshops. Compared to 2008, the workshops HeteroPar (Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms) and XtreemOS Summit (Open Source Grid Operating System) were newcomers, while the workshops SGS (Secure, Trust, Manageable and Controllable Grid Services) and DPA (Abstraction for Distributed Systems) were discontinued in 2009. The following nine workshops were held in conjunction with EuroPar 2009:

– **HPPC 2009** was the 3^{rd} Workshop on Highly Parallel Processing on a Chip. Architectures with a large on-chip parallelism for special and general-purpose computation have been marketed in recent years with various degrees of success. Although the shift toward many-cores is inevitable and generally accepted, the road to travel is still unclear as currently there are many different architectural designs and products by both large and smaller vendors. HPPC 2009 was a forum for discussion of new research directions into parallel (multi-core) architectures, programming models, languages, libraries, algorithms, and software tools. HPCC 2009 started with a well-attended invited talk “The Next 25 Years of Computer Architecture” by Peter Hofstee. It concluded with an interesting and lively panel “Are Many-Core Computer Vendors on Track?” with panelists Uzi Vishkin, Peter Hofstee, Chris Jesshope, and Ahmed Jerraya.

- **HeteroPar 2009** was a workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms. HeteroPar 2009 was the seventh edition of this workshop, but this was the first time that it was co-located with the Euro-Par conference. The workshop intends to be a forum for people working with heterogeneous platforms and trying to find efficient problem solutions on heterogeneous systems. It started with an invited talk by Dick Epema, who discussed different forms of heterogeneity in large-scale systems and the problem of processor co-allocation. HeteroPar 2009 received a large number of submissions and eventually ten papers were accepted for presentation at the workshop.
- **PROPER 2009** was a workshop addressing productivity and performance-tools for HPC application development. The topics covered tools for parallel program development and analysis as well as general performance measurement and visualization. On the one hand, efficient and ease-of-usage of HPC systems is a very important requirement, and on the other hand scalability is essential for achieving high performance. This was the second workshop in co-location with the Euro-Par conference following the success in 2008. The papers presented at the PROPER 2009 workshop discussed performance analysis tools and new design ideas of these tools.
- **ROIA 2009** was the International Workshop on Real-Time Online Interactive Applications on the grid. It was organized in conjunction with the Euro-Par conference for the second time. The workshop intends to cover all areas of real-time distributed technologies, from research of basic real-time methods to applications in real-world environments. In order to promote this research area and to allow more people to become acquainted with ROIA-related applications, the workshop included two tutorial sessions. The papers presented at the workshop discussed the following topics: real-time interactive parallel and distributed tools and environments, real-time interactive distributed (massively multiplayer) online gaming, real-time interactive e-learning applications, integration of Cloud computing virtualization technologies with real-time interactive applications, techniques for real-time quality of service (QoS) monitoring and enforcement, utility business models and service level agreements (SLA) for ROIAs, and experiences in deployment and use of real-world distributed ROIAs.
- **UNICORE Summit 2009** was a unique opportunity for grid users, developers, administrators, and researchers working with the UNICORE grid technology to meet. UNICORE is a well-known grid middleware system which provides a seamless, secure, and intuitive access to distributed grid resources. This was the fifth time the UNICORE Summit was held in conjunction with the Euro-Par conference. The topics discussed by the papers presented at the workshop include: extensions of UNICORE, data management, and deployment.
- **VHPC 2009** was the 4^{th} Workshop on Virtualization in High-Performance Cloud Computing held in conjunction with the Euro-Par conference. The workshop intends to bring commercial providers of cloud computing services and the scientific community together in order to foster discussion,

collaboration, and mutual exchange of knowledge and experience. It provided a good opportunity for the Euro-Par community to get acquainted with this new and very active research domain. The closing session of this year was a panel forum. The discussions at the panel session were very lively and fruitful; the workshop attracted quite a high number of participants.
- **XtreemOS Summit 2009**. XtreemOS is a Linux-based operating systems that includes grid functionalities. It is characterized by properties such as transparency, hiding the complexity of the underlying distributed infrastructure; scalability, supporting hundreds of thousands of nodes and millions of users; and dependability, providing reliability, high availability and security. The XtreemOS summit included presentations and demonstrations about XtreemOS services and components. The workshop has no proceedings.
- **Gecon 2009** was the 6^{th} International Workshop on Grid Economics and Business Models. The commercial exploitation of grid computing is slowly starting to become popular under the term "cloud computing." These existing solutions are very diverse, ranging from resource-as-a-service (RaaS) models to software-as-a-service (SaaS) models. However, the existing cloud offerings can only be purchased directly from a provider; they cannot be traded in a common market. Such a cloud market would act as a focal point for grid buyers and sellers to meet. In fact, in an open market, any market participant could act as a resource provider or resource seller, depending on the current demand level. This approach would allow companies to benefit: on the one hand, excess capacity can be sold to reduce costs; on the other hand, demand peaks can be covered with cheap grid resources. At Gecon 2009 researchers and practitioners from academia and industry were gathered to discuss issues associated with the development of a common market for computing resources, including networks, storage, and software.
- **CoreGRID 2009** was the first workshop organized by the new CoreGRID Working Group sponsored by ERCIM. The CoreGRID workshop has been held in co-location with the EuroPar conference a number of times. Previously, it was organized within the context of the European research Network of Excellence in developing the foundations, software infrastructures, and applications for large-scale, distributed grid and P2P technologies. In 2009 the CoreGRID Working Group was founded after the successful completion of the European Research Network of Excellence project. The CoreGRID Working Group aims to conduct research in the area of the emerging Internet of Services, with direct relevance to the Future Internet Assembly. The grid research community has not only embraced but has also contributed to the development of the service-oriented paradigm to build interoperable grid middleware and to benefit from the progress made by the services research community.

This volume includes the proceedings of the first six workshops; the workshops CoreGRID 2009 and Gecon 2009 have separated proceedings.

The workshops had their own Program Committees and managed their own paper-reviewing process. First of all, we thank all the authors who submitted papers to the various workshops. We are grateful to all the workshop organizers, members of Program Committees, and the many reviewers. Without their contribution organizing the workshops would not have been possible.

Last but not least, we owe a debt of gratitude to the members of the Euro-Par Steering Committee for their support; in particular to Luc Bougé for all his advice regarding organizational issues in workshops. We thank Tomàs Margalef of the organization of Euro-Par 2008 and Eduardo César of the organization of the Euro-Par 2008 workshops for sharing their experience with us. A number of institutional and industrial sponsors contributed toward the organization of Euro-Par 2009. Their names and logos appear on the Euro-Par 2009 website at `http://europar2009.ewi.tudelft.nl/`.

It was our pleasure and honor to organize and host the EuroPar 2009 workshops at Delft University of Technology. We also thank Delft University for the support and facilities they provided during the preparation and during the Euro-Par 2009 workshops.

March 2010

Hai-Xiang Lin
Michael Alexander
Martti Forsell
Adreas Knüpfer
Radu Prodan
Leonel Sousa
Achim Streit

Organization

Euro-Par Steering Commitee

Chair

Chris Lengauer	University of Passau, Germany

Vice-Chair

Luc Bougé	ENS Cachan, France

European Respresentatives

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
Paul Kelly	Imperial College, UK
Harald Kosch	University of Passau, Germany
Thomas Ludwig	University of Heidelberg, Germany
Emilio Luque	Universitat Autònoma of Barcelona, Spain
Tomàs Margalef	Universitat Autònoma of Barcelona, Spain
Wolfgang Nagel	Dresden University of Technology, Germany
Rizos Sakellariou	University of Manchester, UK

Honorary Members

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

Observers

Henk Sips	Delft University of Technology, The Netherlands
Domenico Talia	University of Calabria, Italy

Euro-Par 2009 Organization

Conference Co-chairs

Henk Sips	Delft University of Technology, The Netherlands
Dick Epema	Delft University of Technology, The Netherlands
Hai-Xiang Lin	Delft University of Technology, The Netherlands

Local Organizing Committee

Joke Ammerlaan
Pien Rijnink
Esther van Seters
Laura Zondervan

Web and Technical Support

Stephen van der Laan

Euro-Par 2009 Workshop Program Committees

The Third Workshop on Highly Parallel Processing on a Chip (HPPC 2009)

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
Marc Daumas	University of Perpignan Via Domitia, France
Martti Forsell	VTT, Finland
Peter Hofstee	IBM, USA
Chris Jesshope	University of Amsterdam, The Netherlands
Ben Juurlink	Technical University of Delft, The Netherlands
Jörg Keller	University of Hagen, Germany
Christoph Kessler	University of Linköping, Sweden
Dominique Lavenier	IRISA - CNRS, France
Ville Leppnen	University of Turku, Finland
Radu Marculescu	Carnegie Mellon University, USA
Lasse Natvig	NTNU, Norway
Geppino Pucci	University of Padova, Italy
Jesper Larsson Träff	NEC Laboratories Europe, NEC Europe Ltd., Germany
Uzi Vishkin	University of Maryland, USA

7th Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar 2009)

Steering Committee

Domingo Giménez	University of Murcia, Spain
Alexey Kalinov	Cadence Design Systems, Russia
Alexey Lastovetsky	University College Dublin, Ireland
Yves Robert	Ecole Normale Supérieure de Lyon, France
Denis Trystram	LIG, Grenoble, France

Program Chair

Leonel Sousa	INESC-ID/IST, Technical University of Lisbon, Portugal

Program Committee

Jacques Mohcine Bahi	University of Franche-Comté, France
Mark Baker	University of Reading, UK
Jorge Barbosa	Faculdade de Engenharia do Porto, Portugal
Olivier Beaumont	INRIA Futurs Bordeaux, LABRI, France
Andrea Clematis	IMATI-CNR, Italy
Michel Dayde	IRIT-ENSEEIHT, France
Frédéric Desprez	INRIA, ENS-Lyon, France
Pierre-Franois Dutot	ID-IMAG, France
Toshio Endo	Tokyo Institute of Technology, Japan
Alfredo Goldman	University of São Paulo, Brazil
Abdou Guermouche	University of Bordeaux, France
Shuichi Ichikawa	Toyohashi University of Technology, Japan
Emmanuel Jeannot	INRIA, France
Heleni Karatza	Aristotle University of Thessaloniki, Greece
Tahar Kechadi	University College Dublin, Ireland
Zhiling Lan	Illinois Institute of Technology, USA
Pierre Manneback	Faculté Polytechnique de Mons, Belgium
Satoshi Matsuoka	Tokyo Institute of Technology, Japan
Kiminori Matsuzaki	University of Tokyo, Japan
Nikolay Mirenkov	University of Aizu, Japan
Wahid Nasri	Ecole Sup. des Sciences et Techniques de Tunis, Tunisia
Dana Petcu	University of Timisoara, Romania
Serge Petiton	CNRS/LIFL and INRIA, France
Antonio J. Plaza	University of Extremadura, Spain
Ravi Reddy	University College Dublin, Ireland
Casiano Rodríguez	University of La Laguna, Spain
Mitsuhisa Sato	University of Tsukuba, Japan
Franciszek Seredynski	PJIIT and Polish Academy of Sciences, Poland

H. J. Siegel	Colorado State University, USA
Antonio M. Vidal	Universidad Politecnica de Valencia, Spain
Ramin Yahyapour	University of Dortmund, Germany

Second Workshop on Productivity and Performance Tools for HPC Application Development (PROPER 2009)

Program Chair

Andreas Knüpfer	TU Dresden, Germany

Program Committee

Andreas Knüpfer	TU Dresden, Germany
Jens Doleschal	TU Dresden, Germany
Michael Gerndt	TU München, Germany
Karl Fürlinger	University of California at Berkeley, USA
Allen Malony	University of Oregon, USA
Dieter an Mey	RWTH Aachen, Germany
Shirley Moore	University of Tennessee, USA
Matthias Müller	TU Dresden, Germany
Martin Schulz	Lawrence Livermore National Lab, USA
Felix Wolf	Jülich Supercomputing Centre, Germany

Second International Workshop on Real-time Online Interactive Applications on the Grid (ROIA 2009)

Program Chair

Radu Prodan	Institute for Computer Science, University of Innsbruck, Austria

Program Committee

Christoph Anthes	Johannes Kepler University Linz, Austria
Maximilian Berger	University of Innsbruck, Austria
Marian Bubak	AGH University of Science and Technology, Cracow, Poland / University of Amsterdam, The Netherlands
Frank Glinka	University of Münster, Germany
Sergei Gorlatch	University of Münster, Germany
Petr Holub	Masaryk University, Czech Republic
Alexandru Iosup	Technical University of Delft, The Netherlands
Dieter Kranzlmüller	Ludwig Maximillians-Universität München, Germany
Vlad Nae	University of Innsbruck, Austria
Stefan Podlipnig	University of Innsbruck, Austria
Radu Prodan	University of Innsbruck, Austria

Mike Surridge	IT Innovation Centre, University of Southampton, UK
Jens Volkert	Johannes Kepler University Linz, Austria

UNICORE Summit 2009

Program Chairs

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

Program Committee

Agnes Ansari	CNRS-IDRIS, France
Rosa Badia	Barcelona Supercomputing Center, Spain
Donal Fellows	University of Manchester, UK
Anton Frank	LRZ Munich, Germany
Edgar Gabriel	University of Houston, USA
Alfred Geiger	T-Systems, Germany
Erwin Laure	KTH Stockholm, 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	Fraunhofer Institute SCAI, Germany
Stefan Wesner	University of Stuttgart - HLRS, Germany
Ramin Yahyapour	University of Dortmund, Germany

4th Workshop on Virtualization in High-Performance Cloud Computing (VHPC 2009)

Program Chairs

Michael Alexander (Chair)	scaledinfra technologies GmbH, Austria
Gianluigi Zanetti (Co-chair)	CRS4, Italy

Program Committee

Padmashree Apparao	Intel Corp., USA
Volker Buege	University of Karlsruhe, Germany
Roberto Canonico	University of Naples Federico II, Italy
Tommaso Cucinotta	Scuola Superiore Sant'Anna, Italy
Werner Fischer	Thomas Krenn AG, Germany
William Gardner	University of Guelph, Canada

Table of Contents

Third Workshop on Highly Parallel Processing on a Chip (HPPC 2009)

HPPC 2009: 3rd Workshop on Highly Parallel Processing on a Chip

Martti Forsell[1] and Jesper Larsson Träff[2]

[1] VTT – Technical Research Centre of Finland
Oulu, Finland
Martti.Forsell@vtt.fi
[2] NEC Laboratories Europe, NEC Europe Ltd.
St. Augustin, Germany
traff@it.neclab.eu

Foreword

To counter the relative decline in traditional, single-processor performance, architectures with significant on-chip parallelism for special and general-purpose computation have been marketed in the past few years with various degrees of success. Although the shift toward (general-purpose) parallel processing is inevitable and generally accepted, the road to travel is still essentially unclear – perhaps even unknown, as witnessed by the many different architectural proposals and products by both large and smaller vendors, the usual hype, as well as research projects in many different directions. At present, generally agreed, easily understandable, tractable model architectures, programming models, and programming language concepts to sustain a parallel algorithmics and productive software development (which presupposes a high degree of functional and performance portability) all seem to be missing. The use of highly parallel, special-purpose architectures, which, where applicable, achieve the highest and most efficient performance, is difficult, development intensive, limited in scope, and software development to a large extent suffer from lack of portability. It is obvious that a considerable research, development and teaching effort is called for in the coming decade, and it may well be that the current economics of the software industry cannot be sustained, i.e. that software development may again become seriously labor intensive endeavor.

The Workshop on *Highly Parallel Processing on a Chip* (HPPC) is a forum for presentation and discussion of new research into parallel (multi-core) architectures, programming models, languages, libraries, algorithms, and software tools, including the efficient use of highly parallel special-purpose architectures for general-purpose parallel processing. The workshop especially aims to attract new and tentative work that seriously addresses the problems of managing significant amounts of on-chip parallelism at the levels mentioned. 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. To provide a wider outlook the workshop now regularly features two prominent, opinionated theoretical and practical researchers as invited speakers. To foster discussion between attendees, the invited speakers, and additionally invited panelists, HPPC 2009 featured a (well-visited) panel ("Are many-core computer vendors on track?"). This format will be continued for the next issues of the workshop.

For HPPC 2009, the third installment of the workshop, 5 papers were selected for presentation and subsequent publication out of the 18 received submissions. The submitted papers were all relevant to the workshop themes, some more than others, and due also to the limited time for the workshop, 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 at least *four* reviews (many had five, a few even six) 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. We feel that all papers were given as fair a reading and treatment as is possible these days of severely limited time. The final decision on acceptance was made by the program chairs based on the recommendations from the program committee.

The Euro-Par 2009 workshop day was lively and well-organized, and the HPPC workshop had a high, cumulative attendance of more than 60. In addition to the 5 contributed talks, the workshop had two longer, invited talks by Peter Hofstee (on "The next 25 years of computer architecture?") and Ahmed Jerraya (on "Software Development and Programming of Multi-core SoC"). The HPPC 2009 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 2009 organizers for their help and (excellent) support both before and during the workshop. Our sponsors VTT, NEC Laboratories Europe and Euro-Par 2009 are warmly thanked for the financial support that made it possible to invite Peter Hofstee and Ahmed Jerraya, both of whom we sincerely thank for accepting our invitation to speak and for their excellent talks.

These post-workshop proceedings include the final versions of the presented HPPC papers (as a matter of principle, accepted papers not presented at the workshop will not be 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 has been 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 was introduced with HPPC 2008, and will be continued also for HPPC 2010.

The contributed papers are printed in the order they were presented at the workshop. The abstracts of the two invited talks by Peter Hofstee and Ahmed Jerraya have also been included in the proceedings, as has the panel preamble with five short statements by the panelists. Thematically, the contributed papers cover aspects of multi-core architectures, ("Distance constrained mapping to support NoC platforms based on source routing" by Tornero, Kumar, Mubeen, and Orduña), parallel programming for multi-cores ("Toward metaprogramming for parallel systems on a chip" by Howes, Lokhmotov, Donaldson, and Kelly, "Automatic calibration of performance models on heterogeneous multicore architectures" by Augonnet, Thibault, and Namyst), and use of special-purpose architectures ("Parallel variable-length encoding on GPGPUs" by Balevic, "Dynamic detection of uniform and affine vectors in GPGPU computations" by Collange, Defour, and Zhang).

The HPPC workshop is planned to be organized again in conjunction with Euro-Par 2010.

October 2009 Martti Forsell, VTT, Finland
Jesper Larsson Träff, NEC Laboratories Europe, Germany

The Next 25 Years of Computer Architecture?

Peter Hofstee

IBM Systems and Technology Group
USA
hofstee@us.ibm.com

Abstract. This talk speculates on a technology-driven path computer architecture is likely to have to follow in order to continue to deliver application performance growth over the next 25 years in a cost- and power constrained environment. We try to take into account transistor physics, economic constraints, and discuss how one might go about programming systems that will look quite different from what we are used to today.

Short Biography: H. Peter Hofstee is the IBM Chief Scientist for the Cell Broadband Engine processors used in systems from the Playstation 3 game console to the Roadrunner petaflop supercomputer. He has a masters (doctorandus) degree in theoretical physics from the Rijks Universiteit Groningen, and a PhD in computer science from Caltech. After two years on the faculty at Caltech, Peter joined the IBM Austin research laboratory in 1996 to work on the first GHz CMOS microprocessor. From 2001 to 2008 he worked on Cell processors and was the chief architect of the Synergistic Processor Element.

Software Development and Programming of Multi-core SoC

Ahmed Jerraya

CEA - LETI MINATEC
17, Rue des Martyrs
F-38054 Grenoble Cedex 9, France
ahmed@jerraya@cea.fr

Abstract. SoC designs integrate an increasing number of heterogeneous programmable units (CPU, ASIP and DSP subsystems) and sophisticated communication interconnects. In conventional computers programming is based on an operating system that fully hides the underlying hardware architecture. Unlike classic computers, the design of SoC includes the building of application specific memory architecture and specific interconnect and other kinds of hardware components required to efficiently execute the software for a well-defined class of applications. In this case, the programming model hides both hardware and software interfaces that may include sophisticated communication and synchronization concepts to handle parallel programs running on the processors. When the processors are heterogeneous, multiple software stacks may be required. Additionally, when specific Hardware peripherals are used, the development of Hardware dependent Software (HdS) requires a long, fastidious and error prone development and debug cycle. This talk deals with challenges and opportunities for the design and programming of such complex devices.

Short Biography: Dr. Ahmed Jerraya is Director of Strategic Design Programs at CEA/LETI France. He served as General Chair for the Conference DATE in 2001, Co-founded MPSoC Forum (Multiprocessor System on Chip) and is the organization chair of ESWEEK2009. He supervised 51 PhD's, co-authored 8 Books and published more than 250 papers in International Conferences and Journals.

HPPC 2009 Panel: Are Many-Core Computer Vendors on Track?

Martti Forsell[1], Peter Hofstee[2], Ahmed Jerraya[3], Chris Jesshope[4], Uzi Vishkin[5], and Jesper Larsson Träff[6]

[1] VTT – Technical Research Centre of Finland
Oulu, Finland
[2] IBM Systems and Technology Group
USA
[3] CEA - LETI MINATEC
Grenoble, France
[4] University of Amsterdam
Amsterdam, The Netherlands
[5] University of Maryland Institute of Advanced Computer Studies (UMIACS)
Maryland, USA
[6] NEC Laboratories Europe, NEC Europe Ltd.
St. Augustin, Germany

1 Introduction

The last session of the HPPC 2009 workshop was dedicated to a panel discussion between the invited speakers and three additional, selected panelists. The theme of the panel was originally suggested by Uzi Vishkin, and developed with the moderator. A preamble was given in advance to the five panelists, and provoked an intensive and determined discussion. The panelists were given the chance to briefly summarize their view- and standpoints after the panel.

Panelists: Martti Forsell, Peter Hofstee, Ahmed Jerraya, Chris Jesshope, Uzi Vishkin.

Moderator: Jesper Larsson Träff.

2 Preamble: Background and Issues

The current proliferation of (highly) parallel many-core architectures (homo- and heterogeneous CMP's, GPU's, accelerators) puts an extreme burden on the programmer seeking (or forced) to effectively, efficiently, and with reasonable portability guarantees utilize such processors. The panel will consider whether what many-core vendors are doing now will get us to scalable machines that can be effectively programmed for parallelism by a broad group of users.

Issues that may be addressed by the panelists include (but not exclusively): Will a typical computer science graduate be able to program mainstream, projected many-core architectures? Is there a road to portability between different types of many-core architectures? If not, should the major vendors look for other, perhaps more innovative, approaches to (highly) parallel many-core architectures? What characteristics should such many-core architectures have? Can programming models, parallel languages, libraries, and other software help? Is parallel processing research on track? What will the typical computer science student need in the coming years?

3 Martti Forsell

The sequential computing paradigm formulated in the 50's has been tremendously successful in the history of computing. The main reason for this is the right type of abstraction capturing the essential properties of the underlying machine and providing good portability between machines with substantially different properties. The idealized properties of the computational model – single cycle access time and sequential operation – can be emulated well enough even with speculative superscalar architectures and considerably deep memory hierarchies applying paging virtual memory. At the same time, sequential computing is easy to learn and use, there is a thorough theory of sequential algorithms, and efficient teaching in universities. Synchronization of subtasks of originally parallel computational problems is trivial due to deterministic sequential execution. Performance enhancement techniques, including exploitation of low-level parallelism, speculations, and locality exploitation, are well-known and linked to the paradigm.

Parallel computers introduced in the 60's and the related parallel computing paradigms have had a totally different reception than sequential ones, having doomed them to marginal uses except in high-performance computing. During this decade the situation has, however, changed totally with the arrival of multicore processors. Parallel computing is here to stay with no way back to sequential machines any more. This is because of power density problems preventing exponential growth of clock frequencies for microprocessors. Unfortunately current approaches to parallel computing (e.g. SMP, NUMA, CC-NUMA, vector computing, and message passing) are weak making the whole paradigm poor. Namely, the abstraction of the underlying parallel machinery is too low and inappropriate: A programmer is forced to take care of mapping of functionality, partitioning of data, and synchronization. In the message passing model, one even needs to take care of low-level sending and receiving messages between processes. As a result, programming is difficult and error-prone. The portability between machines with different properties is often limited and easily leads to a need to rewrite the entire software for the new machine. The generality of the theory of parallel algorithms is severely limited by architecture dependency of current solutions, and teaching is virtually nonexistent at elementary level even in universities. Execution of subtasks is asynchronous and the cost of doing an explicit barrier synchronization is easily hundreds or thousands of clock cycles. This severely limits the applicability of current approaches and effectively rules out fine-grained parallel algorithms. Performance enhancement techniques are not particularly innovative nor well-linked to thread-level parallel execution since they are mostly copied from sequential computing, whereas efficient techniques for parallel computing, including co-exploitation of ILP and TLP, concurrent memory access, and multioperations, are missing.

Historically speaking, the trends of architectural approaches towards increasingly parallel and complex machines seem to point towards more difficult programmability. There exists, however, approaches to parallel computing, e.g. vector computing and PRAM, that are easy to program and therefore would solve most of the problems listed above. While the somewhat successful vector computing approach is limited to vectorizable algorithms due to an inability to exploit control parallelism, the more flexible and theoretically beautiful PRAM has been widely considered impossible to implement. According to our implementation estimation studies on advanced parallel architectures this conception appears to be wrong. Therefore, to address programmability and applicability issues, we are developing CMP architectures realizing the PRAM model and related application development methodology. We have just started a project to build our first FPGA prototype. Interestingly we are not alone, two out of five panel participants are doing research in this direction.

4 Peter Hofstee

Driven by the need to deliver continuous performance improvements without disturbing the existing code base, all major high-performance CPU vendors have opted for shared-memory multi-core/multi-thread architectures. With this approach, existing applications with a modest amount of concurrency still benefit from the larger caches, increased memory capacity and bandwidths, and increased I/O capabilities that a next-generation processor provides. The need to provide incremental performance improvements on all applications also is forcing vendors to continue to make modest improvements to per-thread performance, and this limits their ability to achieve the best possible power efficiencies for concurrent applications. All major vendors now integrate the memory controllers. Integration of I/O and graphics is likely to be next leading to more heterogeneous multi core processors. Large machines can be expected to be built as clusters of these SMP nodes, though it is likely that even across these clusters address spaces will be increasingly unified and shared.

Given this type of hardware, the SMP node memory looks more or less “flat”, i.e. access latencies to memory are not significantly dependent on which core on the chip is accessing what memory attached to the node. Even if memory is shared across the cluster, latency and bandwidths are substantially different for memory attached to the local node and memory attached to remote nodes.

In order to prepare students for the future we need a fundamentally new approach to the way students are taught. The fundamentals that drive algorithmic efficiency on today’s and future hardware are:

Classical notions of complexity – The total number of operations (memory accesses, algorithmic operations etc.) as taught today.

Concurrency – A more concurrent algorithm of the same overall complexity is more valuable.

Predictability – An algorithm in which data references and control flow are predictable is more valuable (data parallelism can be regarded as a form of data and control flow predictability).

Locality – An algorithm with better data and control flow locality is more valuable.

Each of these notions of complexity leads to fundamental transformations of the algorithms that are beyond a compiler's ability to perform automatically. Language designers should therefore build on these fundamental notions of algorithmic efficiency while preserving conciseness of expression and semantic clarity. Of course libraries can help those who program computers, not every driver has to be a mechanic, but we should teach computer science students the fundamentals, as we will need many skilled people to restructure our code base such that it can be efficiently targeted at today's and future highly parallel processors.

5 Ahmed Jerraya

The shift from the single processor to heterogeneous multiprocessor architectures poses many challenges for software designers, verification specialists and system integrators. The main design challenges for multi-core processors are: programming models that are required to map application software into effective implementations, the synchronization and control of multiple concurrent tasks on multiple processor cores, debugging across multiple models of computation of MPSoC and the interaction between the system, applications and software views, and the processor configuration and extension.

Programming an MPSoC means to generate software running on the MPSoC efficiently by using the available resources of the architecture for communication and synchronization. This concerns two aspects: software stack generation and validation for the MPSoC and communication mapping on the available hardware communication resources and validation for MPSoC. Efficient programming requires the use of the characteristics of the architecture. For instance, a data exchange between two tasks mapped on different processors may use different schemes through either the shared memory or the local memory of one of these processors. Additionally, different synchronization schemes (polling, interrupts) may be used to coordinate this exchange. Furthermore, the data transfer between the processors can be performed by a DMA engine, thus permitting the CPU to execute other computation, or by the CPU itself. Each of these communication schemes has advantages and disadvantages in terms of performance (latency, throughput), resource sharing (multitasking, parallel I/O) and communication overhead (memory size, execution time). The ideal scheme would be able to produce an efficient software code starting from high-level program using generic communication primitives.

For the design of classic computers, high-level parallel programming concepts (e.g. MPI) are used as an Application Programming Interface (API) to abstract hardware/software interfaces during high level specification of software applications. The application software can be simulated using an execution platform of the API (e.g. MPICH) or executed on existing multiprocessor architectures that include a low level software layer to implement the programming model. In this case the overall performances obtained after hardware/software integration cannot be guaranteed and will depend on the match between the application and the platform. Unlike classic computers, the design of heterogeneous MPSoC requires a better matching between hardware and software in order to meet performances requirements. In this case, the hardware/software interfaces implementation is not standard; it needs to be customized for a specific application in order to get the required performances.

Therefore, for this kind of architectures, classic programming environments do not fit: (i) high level programming does not handle efficiently specific I/O and communication schemes, while (ii) low level programming explicitly managing specific I/O and communication is time consuming and error-prone activity. In practice, programming these heterogeneous architectures is done by developing separate low level codes for the different processors, with late global validation of the overall application with the hardware platform. The validation can be performed only when all the binary software is produced and can be executed on the hardware platform. Next generation programming environments need to combine the high level programming models with the low level details. The different types of processors execute different software stacks. Thus, an additional difficulty is to debug and validate the lower software layers required to fully map the high-level application code on the target heterogeneous architecture.

6 Chris Jesshope

Are manufacturers doing enough is perhaps the wrong question. We should be asking whether they did enough to manage the entirely predictable shift from sequential computing to parallel computing as a direct consequence of the power wall. And the answer is probably no; we are unprepared.

Users have come to expect universality; sequential code works on all architectures either using source-code or binary-code compatibility and this is what they now expect from multi-cores and concurrent heterogeneous systems. Concurrency however, introduces all sorts of difficult problems, including the mapping and scheduling of work, races, deadlocks and fairness issues, etc. Ideally applications engineers (programmers) should not be exposed to the latter.

A key issue therefore is whether we can separate algorithm-engineering issues from concurrency engineering ones. Algorithm engineering does not usually require concurrency, just a deterministic parallel implementation. A major problem is in dealing with synchronisation state, it complicates algorithm engineering unnecessarily and constrains the problem mapping. It is not strictly necessary and there are approaches, which aim to provide such a separation of concerns. These are emerging technologies however, and are academic rather than commercial.

A further issue is whether we can program in a manner which is independent of the scale of concurrency to which the the code will eventually be targeted, i.e. can we code once and run anywhere, in order to have code portability across different classes of target architecture. Again there seems to have been little work moving us in this direction. It requires abstract concurrent programming models that allow the capture of maximal concurrency and, ideally, also capture locality. A typical approach is to take code and to parallelise it to given target with a given granularity of parallelism. However, when you change the parameters or the target it needs to be rewritten. The alternative is to program at the finest grain possible and then sequentialise the code automatically when a target is chosen. In this way the same code can be efficiently executed on any target and the procedure of sequencing parallelism is a rather trivial one compared to parallelisation. Again work is being carried out in this area but is also academic.

Models that are maximally concurrent but abstract (e.g. SVP) and coordination languages that allow this separation of concerns (e.g. S-Net) have been developed in the EU AETHER project, which has taken a 10-year-out view on programming highly concurrent and heterogeneous systems (see: `http://www.aether-ist.org/`).

7 Uzi Vishkin

Hardware vendors have been forced into replacing the serial paradigm that has worked for them well for decades by parallel computing based on many-core architectures. To date, no commercial easy-to-program general-purpose many-core machine for single-task completion-time has been available. In fact, several decades of parallel architectures have been able to produce only rather limited success stories. A 2003 NSF Blue Ribbon committee effectively declared their programming a "disaster area" by noting that to many programmers it is "as intimidating and time consuming as programming in assembly language". Hardware vendors need to reproduce the serial success for many-cores. Adopting, without significant modification, the same parallel architectures would instead drag mainstream computing into the same disaster area.

Customers buying a computer interact with its software, but their link to the hardware is indirect, by nature. However, the cyclic process of hardware improvements leading to software improvements, which lead back to hardware improvements and so on, known as the software spiral, facilitated for many years a direct link between customers and hardware. Hardware designers could directly serve their customers by helping to run serial code faster. Alas, the software spiral is now broken. Consequently, getting application software developers (ASDs) to switch to the emerging generation of many-core systems has become much more critical to serving these customers. However, the incentive to develop software for the new machines has decreased considerably. Code development and maintenance is much more expensive, as initial development time is higher and code is more error prone. Not only that the investment is higher, the returns on it are much riskier: even if machines continue to support the current development platform, some hard-to-predict future upgrades may offer new options for optimization of performance, allowing competitors to develop better software, at a lesser cost, by just adopting a wait-and-see approach. Thus, computer designers need to understand the legitimate concerns of software developers and do what they can to "woo" them.

The explicit multi-threading (XMT) approach `www.umiacs.umd.edu/users/vishkin/XMT/` could affect the above discussion in two ways. First, it affirms concerns that hardware improvements that may significantly reduce investment in code development by just waiting till they are installed are indeed possible. The second way is that incorporation of the hardware upgrades that XMT suggests, could make it possible to support the broad family of PRAM algorithms, greatly alleviating current concerns about ease-of-programming. Moreover, every person majoring in CS should be able to program the commodity many-core system of the future. Teachability of XMT programming has been demonstrated at various levels from rising 6th graders to graduate students, and students in a freshman class were able to program 3 parallel sorting algorithms.

Distance Constrained Mapping to Support NoC Platforms Based on Source Routing[⋆]

Rafael Tornero[1], Shashi Kumar[2], Saad Mubeen[2], and Juan Manuel Orduña[1]

[1] Departamento de Informática, Universitat de València, Spain
{Rafael.Tornero,Juan.Orduna}@uv.es
[2] School of Engineering, Jönköping University, Sweden
{Shashi.Kumar,mems07musa}@jth.hj.se

Abstract. Efficient NoC is crucial for communication among processing elements in a highly parallel processing systems on chip. Mapping cores to slots in a NoC platform and designing efficient routing algorithms are two key problems in NoC design. Source routing offers major advantages over distributed routing especially for regular topology NoC platforms. But it suffers from a serious drawback of overhead since it requires whole communication path to be stored in every packet header. In this paper, we present a core mapping technique which helps to achieve a mapping with the constraint over the path length. We have found that the path length constraint of just 50% is sufficient in most cases. We also present a method to efficiently compute paths for source routing leading to good traffic distribution. Evaluation results show that performance degradation due to path length constraint is negligible at low as well as high communication traffic.

Keywords: Network on Chip, Core Mapping, Routing Algorithms, Source Routing.

1 Introduction

As Semi-conductor Technology advances, it becomes possible to integrate a large number of Intellectual Property (IP) cores, like DSPs, CPUs, Memories, etc, on a single chip to make products with complex and powerful functions. Efficient communication infrastructure is crucial for harnessing enormous computing power available on these Systems on Chip (SoCs). Network on Chip (NoC) is being considered as the most suitable candidate for this job [1].

Many design choices and aspects need to be considered for designing a SoC using NoC paradigm. These include: network topology selection, routing strategy selection and application mapping. Both application mapping and routing strategy have big impact on the performance of the application running on a NoC platform. The application mapping problem consist of three tasks: i) the

[⋆] This work has been jointly supported by the Spanish MEC and European Commission FEDER funds and the University of Valencia under grants Consolider Ingenio-2010 CSD2006-00046 and TIN2009-14475-C04-04 and V-SEGLES-PIE Program.

application is split into a set of communication tasks, generally represented as a task graph; ii) the tasks are assigned and scheduled on a set of IP cores selected from a library; iii) the IP cores have to be placed onto the network topology in such a way that the desired metrics of interest are optimized. The first two steps are not new since they have been extensively addressed in the area of hardware/software co-design and IP reuse [2] by the CAD community. The third step, called topological mapping, has recently been addressed by a few research groups [3], [4].

One way to classify the routing algorithms is by considering the component in the network where the routing decision to select the path is done. Under this consideration the routing algorithms are classified into source routing and distributed routing algorithms. In source routing algorithms, the path between each pair of communicating nodes is computed offline and stored at each source node. When a core needs to communicate with another core the encoded path information is put in the header of each packet. In distributed routing, the header only needs to carry the destination address and each router in the network has competence to make the routing decision based on the destination address.

Source routing has not been considered so far for NoCs due to its perceived underutilization of network bandwidth due to the requirement of large number of bits in the packet header to store path information. This conclusion may be valid perhaps for large dynamic networks where network size and topology are changing. But in a NoC with fixed and regular topology like mesh, the path information can be efficiently encoded with small number of bits. Saad et. al. [5] have made a good case for use of source routing for mesh topology NoCs. It can be easily shown [6] that two bits are sufficient to encode information about one hop in the path. Since the packet entering a router contains the pre-computed decision about the output port, the router design is significantly simplified. Also, since NoCs used in embedded systems are expected to be application specific, we can have a good profile of the communication traffic in the network [7]. This allows us to offline analyze the traffic and compute efficient paths according to the desired performance characteristics, like uniform traffic load distribution, reserved paths for guaranteed throughput etc.

Figure 1 shows an application, that has been assigned and scheduled on eight cores, topologically mapped on a 4x2 mesh. The Application Characterization Graph (APCG) of this application can be seen in Figure 1(a), where a node corresponds to a core and a directed edge corresponds to communication between two connected cores. APCG will be defined more formally in section 2. Assuming minimal routing, the maximum route length required is equal to the diameter of the topology. It means that, if the diameter was used for this example, the required path length would be 4 hops and therefore 10 bits would be required to code a route (see Figure 1(b)). However, it is possible to find a mapping in which the maximum distance between two communicating cores is much smaller than the diameter. Figure 1(c) shows such a mapping for the example in which the maximum distance is just two hops and only 6 bits are required for the path information.

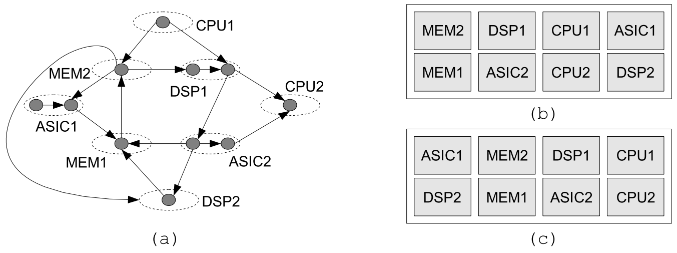


Fig. 1. Different mappings of the same application. a) the APCG, b) A distance unconstrained mapping, c) A 2-hops constrained mapping.

Close compactness of mapping could lead to higher congestion in certain links. Since routers for source routing are relatively faster than routers for distributed routing, the above disadvantage will be adequately compensated [6].

Related Work
A large number of routing algorithms have been proposed in literature for NoCs. Most proposals fall in the category of distributed adaptive routing algorithms and provide partial adaptivity thus providing more than one path for most communicating pairs and at the same time avoiding possibility of deadlocks. In [7], Palesi et al. propose a methodology to compute deadlock free routing algorithms for application specific NoCs with the goal of maximizing the communication adaptivity.

Several works have been proposed in literature in the context of core mapping. Hu et al. present a branch and bound algorithm for mapping cores in a mesh topology NoC architecture that minimizes the total amount of power consumed in communications [8]. Murali et al. present a work to solve the mapping problem under bandwidth constraint with the aim of minimizing the communication delay by exploiting the possibility of splitting the traffic among various paths [3]. Hansson et al. present an unified single objective algorithm which couples path selection, mapping of cores, and channel time-slot allocation to minimize the network required to meet the constraints of the application [9]. Tornero et al. present a communication-aware topological mapping technique that, based on the experimental correlation of the network model with the actual network performance, avoids the need to experimentally evaluate each mapping explored [4]. In [10], Tornero et al. present a multi-objective strategy for concurrent mapping and routing for NoC. All the aforementioned works address the integration of mapping and routing concurrently but taking into account only the distributed routing functions.

Although source routing has been shown to be efficient for general networks [11], it had not been explored so far for NoC architectures. Recently Mubeen et.al [5] have made a strong case for source routing for small size mesh topology NoCs. The author, has demonstrated that source routing can have higher communication performance than adaptive distributed routing [6]. In this paper we modify our earlier approach and tackle the integration of topological mapping for NoC platforms which use limited path length source routing for inter-core communication.

2 Problem Formulation

Simply stated, our goal is to find an arrangement of cores in tiles together with path selection such that the global communication cost is minimized and the maximum distance among communicating cores is within the given threshold. The value of the threshold comes from the fixed on-chip communication infrastructure (NoC) platform which uses source routing with an upper limit on the length of the path. Before formally defining the problem, we need to introduce the following definitions [8].

Definition 1. An Application Characterization Graph $APCG = G(C, A)$ is a directed graph, where each vertex $c_i \in C$ represents a selected IP core, and each directed arc $a \in A$ characterizes the communication process from core c_i to core c_j. For each communication $a \in A \wedge a = (c_i, c_j)$, the function $B(a)$ returns the bandwith requirements of a. This is minimum bandwith that should be allocated by the network in order to meet the performance constraint for communication a.

Definition 2. An Architecture Characterization Graph $ARCG = G(T, L)$ is a directed graph which models the network topology. Each vertex t_i represents a tile, and each directed arc l_{ij} represents the channel from tile t_i to tile t_j.

We must solve two problems: first, we have to find a mapping within the constraint of maximum distance allowed by the communication platform. The second problem to be solved is to compute efficient paths for all communicating pairs of cores such that there is no possibility of deadlock as well as the traffic is well balanced. We can formulate the first problem as follows. Given the APCG and the ARCG, that satisfy $|C| \leqslant |T|$, find a mapping function M from C to T which minimizes the mapping cost function M_c:

$$\min\{M_c = \sum_{\substack{\forall c_i, c_j \in C \\ a=(c_i,c_j) \in A}} B(a) * (dist(M(c_i), M(c_j)))^3\} \tag{1}$$

such that:

$$dist(M(c_i), M(c_j)) \leq Threshold \wedge a = (c_i, c_j) \in A \ . \tag{2}$$

In the equation 1, the second term of the summation is raised to power 3 with the aim of giving more importance to the distance in the search for a pseudo-optimal mapping. This value is a trade-off between the quality of the results and the computation time (the power 2 provides poor quality of results and power of 4 and higher ones are too much time consuming). Condition 2 guarantees that every pair of communicating cores should be mapped such that the Manhattan distance between them is less than the **threshold**. We assume that the underlying source routing uses only minimal distances. Nevertheless, this **threshold** cannot be smaller than the lower bound on the path length required for mapping APCG on ARCG. The lower bound in this context refers to a value such that any possible mapping will have at least one pair with distance more than or equal to the lower bound. For example, the lower bound for the APCG in our example is 2, since there is no possibility to find a mapping with distance 1. The APCG together with the ARCG can be analyzed in order to find the lower bound for the mapping.

In a 4x2 mesh topology, a node can be connected to maximum three other cores with a distance 1. A core can be connected to up to 6 cores if distance of 2 hops is allowed. There may not be any 2 distance constrained mapping available for an APCG with maximum out-degree 5. If L is the lower bound, then one can start by searching for a mapping with constraint equal to L. If one fails then one must repeat the process of finding a feasible mapping by using $L + 1, L + 2, \ldots$ and so on as the constraint.

Once the cores are mapped satisfying (2), the second problem is to find a path for every communicating core pair C_i and C_j such that: the path length is equal to Manhattan distance between C_i and C_j; there is no possibility of deadlock when some or all other core pairs communicate concurrently and the traffic load on all the links in the network is as balanced as possible.

3 The Distance Constrained Mapping Algorithm

We have modified our earlier mapping approach developed for NoC platforms using distributed routing techniques [4] to obtain distance constrained topological mapping. This approach considers the network resources and the communication pattern generated by the tasks assigned to different cores in order to map such cores to the network nodes. The method is based on three main steps.

Step 1. Model the network as a table of distances (or costs) between each pair of source/destination nodes. The cost for communicating each pair of nodes is computed as inversely related to the available network bandwidth.
Step 2. Perform a heuristic search in the solution space with the aim of obtaining a near-optimal mapping that satisfies our distance restrictions.
Step 3. Repair the mapping found in the second step if some communicating pairs violate the distance constraints.

We have computed steps 1 and 2 as in our previous work ([4]). If the mapping found by the heuristic method does not satisfy the distance constraint, then a heuristic repair procedure, step 3, tries to repair the solution found. This procedure, based on [12], consists of a hill-climbing that minimizes the number of constraints violated by the solution mapping. The advantage of this procedure

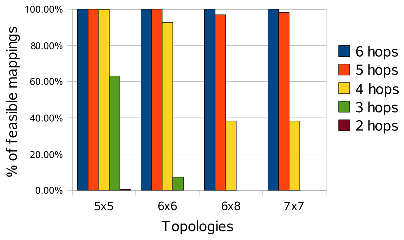


Fig. 2. Feasible mappings with distant constraint

is that it is fast to compute, but presents some drawbacks like the capacity to fall in a local minimum that does not satisfy the distance constraint. This drawback could be reduced by repeating the same ï¿½procedure several times.

It should be noticed that the goal of this work is to prove that it is possible to obtain efficient solutions for the problem of distance constrained mapping by using a mapping technique analogous to the one shown in our previous work ([4]). In order to achieve this goal, we have used the same heuristic method for solving step 2 than the one shown in [4]. Since that method was not designed for this problem, a new heuristic method specifically developed for solving this problem is likely to provide better solutions.

3.1 Feasibility Experiments

In order to test the distance constrained topological mapping method we have made a set of feasibility experiments. These consists of mapping 500 random APCGs on several 2-D mesh topologies. We have used a $5x5$, $6x6$, $6x8$ and $7x7$ mesh topologies. Each node of each APCG presents an out-degree log-normally distributed with a mean of 2 and a standard deviation of 1 communications. This distribution was motivated by analysis of multi-media applications used in literature [7,8]. We have used an uniform probability distribution for spatial communications. It means that the probability of a core c_i communicating with a core c_j is the same for every core. The communication bandwith between each pair of communicating cores is distributed uniformly between 10 and 100 Kbytes/sec.

Figure 2 shows the result of the experiments. The X-axis shows the topologies and the Y-axis shows the percentage of feasible mappings. Each bar in the figure presents for each topology the percentage number of APCGs the method is able to map given a distance. As can be seen, 96% of the cases, the mapping technique is able to map the 500 APCGs with a distance of only 5 hops for all the topologies tried. It means that the path length of the header flit[1] can be reduced from the diameter of the topology to 5 hops reducing the network overhead.

[1] Flit stands for flow control digit and represents the unit of data that is transmitted over a logical link (channel) in a single step.

Table 1. Best routing algorithm for a traffic type

Traffic Type	Best Routing Algorithm
Random Traffic	XY
Hot-Spot Traffic	Odd-Even
East-Dominated Traffic	West First
West-Dominated Traffic	East First
Transpose Traffic	Negative First

4 Efficient Path Computation for Source Routing

After the cores have been mapped on the NoC platform which supports distance constrained source routing, the next step will be to compute efficient paths for all the communicating pairs of cores. For each source core these paths will be stored as a table in the corresponding resource (core) to network interface (RNI). The RNI will use this table to append the path in the header flit of the packet. Beside avoiding deadlocks, the computed paths should also avoid congestion and uniformly distribute traffic among the links in the network as much as possible.

4.1 Routing Algorithm Selection

A large number of deterministic and adaptive routing algorithms are available for deadlock free routing in mesh topology NoCs. The most famous among these are XY, Odd-Even, West First and North Last routing algorithms. XY is a deterministic routing algorithm and allows a single path between every pair of nodes. The other algorithms are partially adaptive routing algorithms and prohibit the packets to take certain turns, but allow path adaptivity for most pairs. It has been shown that no single routing algorithm provides best performance for all types of traffic. Table 1 gives relatively best routing algorithm for some specific types of communication traffic [6].

A traffic is called West-Dominated if majority of communication (considering number of communications and communication volume) is from east to west. We analyze and classify the traffic using the mapped APCG and select the most appropriate routing algorithm. The analysis uses the relative position of source and destination cores and the communication volume between pairs [6].

4.2 Path Computation

One can easily compute a path for each communication pair using the routing algorithm selected using the analysis in the previous subsection. In the case of deterministic routing the only path available is selected. In the case of partially adaptive routing algorithm the path is constructed by making a choice with a uniform probability at all intermediate routers where a choice among multiple admissible ports is available. Our study has shown that communication traffic

```
Algorithm Path-Computations($RA$, $C$, $P$)
 /* Implicit inputs: Mappings of cores to slots in topology */
 /* Inputs: $RA$ - Routing Algorithm, $C$ - Set of Communications */
 /* Outputs: Set of paths $P = \{Pi,\ i = 1..N,\ Pi\ used\ for\ Ci\}$ */
 begin
   1. Initialize load on each link $li = 0,\ i = 1..Number\ of\ links$
   2. Order communications $Ci,\ i = 1..N$ in ascending order based on
      communication volume
   3. For $i = 1$ to $N$ do
        - Find a path $Pi$ for $Ci$ using $RA$ considering
          current loads on various links and update the loads
 end
```

Fig. 3. Pseudocode of the algorithm for path computation

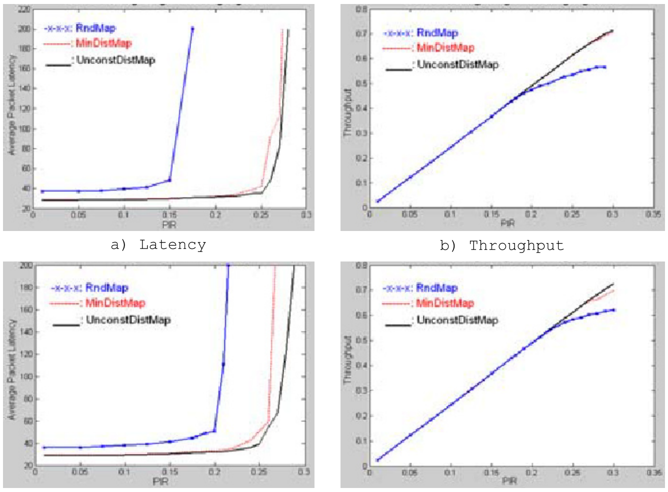
type is rarely pure. For example, it is rare to have an application with pure West-Dominated traffic. To handle this we use adaptivity of the routing algorithm to balance load on the links and avoid/reduce congestion. Figure 3 describes a constructive algorithm to achieve this.

All the communications are sorted in ascending order according to their communication volume. Then in each iteration a path is computed for one communication using the selected routing algorithm. At every router where there exists a choice among multiple output ports, the port is selected if a lower load is already mapped to the corresponding output link. We keep updating the estimated load on links after each iteration. It has been shown that this methodology leads to efficient paths for communication [6].

5 Evaluation and Results

For evaluation purposes, we have evaluated the proposed approach using a set of random traffic scenarios. Each traffic scenario has been generated as described in section 3.1. For each scenario we have computed a random mapping, a near-optimal unconstrained mapping and a near-optimal constrained mapping for 5 hops. The Figure 4 shows the performance results in terms of latency and throughput for two of such traffic scenarios. In this figure the random mapping, the near-optimal unconstrained mapping and the near-optimal distance constrained mapping have been labeled as RNDMAP, UNCONSTDISTMAP and MINDISTMAP respectively.

The performance evaluation has been carried out using a NoC simulator developed in SDL language [6]. The simulator implements a NoC Platform based on a 2-D $7x7$ mesh topology and source routing. The simulated NoC also uses wormhole switching with a packet size fixed to 10 flits, the input buffers have capacity for keep 4 flits. We have used the source packet injection rate (pir) as load parameter. A Matlab based tool has been developed to compute efficient paths for source routing as described in Section 4. For each load value, latency and throughput values are averaged over 20,000 packets drained after a warm-up of 2000 packets drained.



a) Latency

b) Throughput

c) Latency

d) Throughput

Fig. 4. Simulation results for two random APCGs

Figure 4(a) and Figure 4(c) show the simulation results for average latency in cycles. These figures show that at both low traffic loads and high traffic loads both the UNCONSTDISTMAP and MINDISTMAP present similar behaviour and save more than 20% cycles and 25% over RNDMAP respectively.

The throughput results, measured in packets/cycle, are shown in Figure 4(b) and Figure 4(d). As we can look at the throughput achieved by the MINDISTMAP is almost the same as the throughput achieved by the UNCONSTDISTMAP and much higher than the throughput achieved by a RNDMAP close to saturation.

Therefore, the evaluation results show that is possible reduce the path length of the header flit at least to half of the network diameter without significantly degradation of the performance.

6 Conclusions and Future Work

We have addressed the application mapping problem for NoCs when the communication infrastructure is pre-designed as 2-D mesh topology using source routing and the path length header field is delimited up by a number of hops significantly less than the network diameter. In such a scenario we have demonstrated that our distance constrained mapping technique is able to map 96% of applications tried with a distance constraint less than half of the network diameter. We have proposed an efficient method, based on existing distributed deadlock free routing algorithms, to compute efficient paths required for source routing. The simulation based evaluation results show that our distance constrained mapping gives more than 20% latency improvement over random mapping at low traffic loads. The saturation points traffic load also shows around 25% . Performance degradation as compared to path length constraint is negligible at low communication traffic and saturation packet injection rate is also only reduced by just 5%.

To the best of our knowledge this is the first attempt to consider core mapping for NoC platforms based on source routing. As future work, we plan to use intelligent heuristic search method to further lower the path length thus reducing bandwidth underutilization of NoC. We are also working on methods to improve computed paths for better link load balancing. Testing this approach on large realistic applications will also demonstrate the feasibility of our approach.

References

1. Benini, L., De Micheli, G.: Networks on chips: a new soc paradigm. Computer 35(1), 70–78 (2002)
2. Chang, J.M., Pedram, M.: Codex-dp: co-design of communicating systems using dynamic programming. In: Proc. Design Automation and Test in Europe Conference and Exhibition 1999, March 9-12, pp. 568–573 (1999)
3. Murali, S., De Micheli, G.: Bandwidth-constrained mapping of cores onto noc architectures. In: Proc. Design, Automation and Test in Europe Conference and Exhibition, February 16-20, vol. 2, pp. 896–901 (2004)
4. Tornero, R., Orduña, J.M., Palesi, M., Duato, J.: A communication-aware topological mapping technique for nocs. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 910–919. Springer, Heidelberg (2008)
5. Mubeen, S., Kumar, S.: On source routing for mesh topology network on chip. In: SSoCC'09: 9th Swedish System on Chip Conference (May 2009)
6. Mubeen, S.: Evaluation of source routing for mesh topology network on chip platforms. Master's thesis, School of Engineering, Jönköping University (2009)
7. Palesi, M., Holsmark, R., Kumar, S., Catania, V.: Application specific routing algorithms for networks on chip. IEEE Transactions on Parallel and Distributed Systems 20(3), 316–330 (2009)
8. Hu, J., Marculescu, R.: Energy-aware mapping for tile-based noc architectures under performance constraints. In: ASPDAC: Proceedings of conference on Asia South Pacific Design Automation, pp. 233–239. ACM, New York (2003)
9. Goossens, K., Radulescu, A., Hansson, A.: A unified approach to constrained mapping and routing on network-on-chip architectures. In: Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '05, September 2005, pp. 75–80 (2005)
10. Tornero, R., Sterrantino, V., Palesi, M., Orduña, J.M.: A multi-objective strategy for concurrent mapping and routing in network on chip. In: Proc. of the 23rd IEEE International Parallel and Distributed Processing Symposium, May 25-29 (2009)
11. Flich, J., López, P., Malumbres, M.P., Duato, J.: Improving the performance of regular networks with source routing. In: Proceedings of the 2000 International Conference on Parallel Processing, p. 353. IEEE Computer Society, Los Alamitos (2000)
12. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method. In: AAAI, pp. 17–24 (1990)

Parallel Variable-Length Encoding on GPGPUs

Ana Balevic

University of Stuttgart
ana.balevic@gmail.com

Abstract. Variable-Length Encoding (VLE) is a process of reducing input data size by replacing fixed-length data words with codewords of shorter length. As VLE is one of the main building blocks in systems for multimedia compression, its efficient implementation is essential. The massively parallel architecture of modern general purpose graphics processing units (GPGPUs) has been successfully used for acceleration of inherently parallel compression blocks, such as image transforms and motion estimation. On the other hand, VLE is an inherently serial process due to the requirement of writing a variable number of bits for each codeword to the compressed data stream. The introduction of the atomic operations on the latest GPGPUs enables writing to the output memory locations by many threads in parallel. We present a novel data parallel algorithm for variable length encoding using atomic operations, which archives performance speedups of up to 35-50x using a CUDA-enabled GPGPU.

1 Introduction

Variable-Length Encoding (VLE) is a general name for compression methods that take advantage of the fact that frequently occurring symbols can be represented by shorter codewords. A well known example of VLE, Huffman coding [1], constructs optimal prefix codewords on the basis of symbol probabilities, and then replaces the original symbols in the input data stream with the corresponding codewords.

The VLE algorithm is serial in nature due to data dependencies in computing the destination memory locations for the encoded data. Implementation of a variable length encoder on a parallel architecture is faced by the challenge of dealing with race conditions when writing the codewords to a compressed data stream. Since memory is accessed in fixed amounts of bits whereas codewords have arbitrary bit size, the boundaries between adjacent codewords do not coincide with the boundaries of adjacent memory locations. The race conditions would occur when adjacent codewords are written to the same memory location by different threads. This creates two major challenges for creating a parallel implementation of VLE: 1) computing destination locations for the encoded data elements with a bit-level precision in parallel and 2) managing concurrent writes of codewords to destination memory locations.

In recent years, GPUs evolved from simple graphics processing units to massively parallel architectures suitable for general purpose computation, also known as GPGPUs. The NVIDIA GeForce GTX280 GPGPU used for this paper provides 240 processor cores and supports execution of more than 30,000 threads at once. In image and video processing, GPGPUs have been used predominantly for the acceleration of inherently data-parallel functions, such as image transforms and motion estimation algorithms [2,3,4]. The VLE entropy coding to our best knowledge has not been implemented on GPUs so far, due to its inherently serial nature. Some practical compression-oriented approaches on GPUs include compaction and texture compression. The compaction is a method for removing unwanted elements from the resulting data stream by using the parallel prefix sum primitive [5]. An efficient implementation of the stream reduction for traditional GPUs can be found in [6]. The texture compression is a fixed-ratio compression scheme which replaces several pixels by one value. Although it has a fast CUDA implementation [7], it is not suitable for codecs requiring a final lossless encoding pass, since it introduces a loss of fidelity.

We propose a fine-grain data parallel algorithm for lossless compression, and present its practical implementation on GPGPUs. The paper is organized as follows: Section 2 gives an overview of GPGPU architecture, in Section 3 we present a design and implementation of a novel parallel algorithm for variable-length encoding (PAVLE), and in Section 4, we present performance results and discuss effects of different optimizations.

2 GPGPU Architecture

The unified GPGPU architecture is based on a parallel array of programmable processors [8]. It is structured as a set of multiprocessors, where each multiprocessor is composed of a set of simple processing elements working in SIMD mode. In contrast to CPU architectures which rely on multilevel caches to overcome long system memory latency, GPGPUs use fine-grained multi-threading and a very large number of threads to hide the memory latency. While some threads might be waiting on data to be loaded from the memory, the fine-grain scheduling mechanism ensures that ready warps of threads (scheduling unit) are executed, thus providing effectively highly parallel computation resources.

The memory hierarchy of the GPGPU is composed of global memory (high-latency DRAM on the GPU board), shared memory and register file (low-latency on-chip memory). The logical organization is as follows: the global memory can be accessed among all the threads running on the GPU without any restrictions; the shared memory is partitioned and each block of threads can be assigned one exclusive partition of the shared memory, and the registers are private to each thread. When GPU is used as a coprocessor, the data needs to be transferred first from the main memory of host PC to the global memory. In this paper, we will assume that the input data is located in the global memory, e.g. as a result of a computation or explicit data transfer from the PC.

The recent Tesla GPGPU architectures introduce hardware support for atomic operations. The atomic operations provide a simple method for safely handling race conditions, which occur when several parallel threads try to access and modify data at the same memory location, since it is guaranteed that if an atomic instruction executed by a warp reads, modifies, and writes to the same location in global memory for more than one of the threads of the warp, each access to that memory location will occur and will be serialized, but the order in which they occur is not defined [9]. The CUDA 1.1+ GPU devices support the atomic operations on 32-bit and 64-bit words in the global memory, while CUDA 1.3 also introduces support for shared memory atomic operations.

3 The Parallel Variable-Length Encoding Algorithm

This section presents the parallel VLE (PAVLE) algorithm for GPGPUs with hardware support for atomic operations. The parallel variable-length encoding consists of the following parallel steps: (1) assignment of codewords to the source data, (2) calculation of the output bit positions for compressed data (codewords), and finally (3) writing (storing) codewords to the compressed data array. A high-level block-diagram of the PAVLE encoder is given in Fig. 1. Pseudocode for the

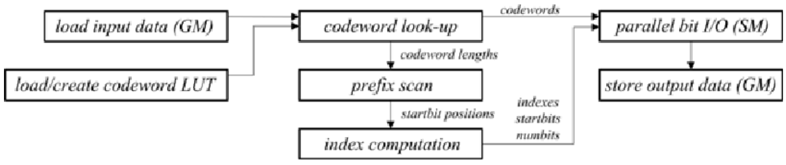


Fig. 1. Block diagram of PAVLE algorithm

parallel VLE is given in Listing 1 with lines 2 - 5 representing the step 1, lines 6 - 8 being the step 2 and lines 9 - 28 representing the step 3. The algorithm can be simplified if one assumes a maximal codeword length, as is done in the case for the JPEG coding standard. Restricting the codeword size reduces the number of control dependencies and also reduces the amount of temporary storage required, resulting in much greater kernel efficiency.

3.1 Codeword Assignment to Source Data

In the first step, variable-length codewords are assigned to the source data. The codewords can be either computed using an algorithm such as Huffman [10], or they can be predefined, e.g. as it is frequently the case in image compression implementations. Without loss of generality, we can assume that the codewords are available and stored in a table. This structure will be denoted as the codeword look-up table (codeword LUT). Each entry in the table contains two values: the binary code for the codeword, and codeword length in bits, denoted as a *(cw, cwlen)* pair. Our implementation uses an encoding alphabet of up to 256 symbols, with each symbol representing one byte. During compression, each source data symbol (byte) is replaced with the corresponding variable-length codeword.

The PAVLE is designed in a highly parallel manner, with one thread processing one data element. The threads load source data elements and perform codeword look-up in parallel. As the current GPGPU architecture provides more efficient support for 32-bit data types, the source data is loaded as 32-bit unsigned integers to shared memory, where it is processed by blocks of threads. The 32-bit data values are split into four byte symbols, which are then assigned corresponding variable-length codewords from the codeword LUT. The codewords are locally concatenated into an aggregate codeword, and the total length of the codeword in bits is computed.

Algorithm 1. Parallel Variable Length Encoding Algorithm

```
 1: $k \leftarrow tid$
 2: for threads $k = 1$ to $N$ in parallel
 3:     $symbol \leftarrow data[k]$
 4:     $cw[k], cwlen[k] \leftarrow cwtable[symbol]$
 5: end for
 6: for threads $k = 1$ to $N$ in parallel
 7:     $bitpos[1..N] \leftarrow prefixsum(cwlen[1..N])$
 8: end for
 9: for threads $k = 1$ to $N$ in parallel
10:     $kc \leftarrow bitpos[k]$ div $ws$
11:     $startbit \leftarrow bitpos[k]$ mod $ws$
12:     while $cwlen[k] > 0$ do
13:         $numbits \leftarrow cwlen[k]$
14:         $cwpart \leftarrow cw[k]$
15:         if $startbit + cwlen > wordsize$ then
16:             $overflow \leftarrow 1$
17:             $numbits \leftarrow wordsize - startbit$
18:             $cwpart \leftarrow$ first $numbits$ of $cw[k]$
19:         end if
20:         $put\_bits\_atomic(out, kc, startbit, numbits, cwpart)$
21:         if $overflow$ then
22:             $kc \leftarrow kc + 1$
23:             $startbit \leftarrow (startbit + numbits)$ mod $wordsize$
24:             remove first $numbits$ from $cw[k]$
25:             $cwlen[k] \leftarrow cwlen[k] - numbits$
26:         end if
27:     end while
28: end for
```

3.2 Computation of the Output Positions

To store the data which does not necessarily match the size of addressable memory locations, it is necessary to compute the destination address in the memory and also the starting bit position inside the memory location. Since in the previous parallel step the codewords were assigned to input data symbols, the dependency in computation of the codeword output locations can be resolved based on the knowledge of the codeword lengths. The output parameters for each codeword are determined by computing the number of bits that should precede each codeword in the destination memory. The bit offset of each codeword is computed as a sum of assigned codeword lengths of all symbols that precede that element in the source data array. This can be done efficiently in parallel by using a prefix sum computation.

The prefix sum is defined in terms of a binary, associative operator $+$. The prefix sum computation takes as input a sequence $x_0, x_1, ..., x_{n-1}$ and produces an output sequence $y_0, y_1, ..., y_{n-1}$ such that $y_0 = 0$ and $y_k = x_0 + x_1 + ... + x_{k-1}$. We use a data-parallel prefix sum primitive [11] to compute the sequence of output bit offsets y_k on the basis of codeword lengths x_k, that were assigned to source data symbols. A work-efficient implementation of parallel prefix sum performs $O(n)$ operations in $O(\log n)$ parallel steps, and it is the asymptotically most significant component in the algorithmic complexity of the PAVLE algorithm. Given the bit positions at which each codeword should start in the compressed

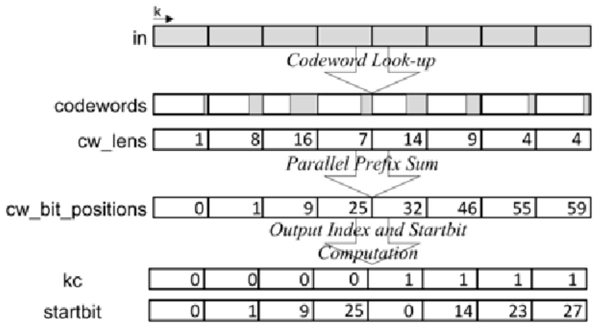


Fig. 2. An example of the variable-length encoding algorithm

data array in memory, the output parameters can be computed knowing the fixed machine word size, as given in the lines 10-11 of the pseudocode.It is assumed that the size of addressable memory locations is 32-bits, and it is denoted as *wordsize*. The variable k is used to denote the unique thread Id. It also corresponds to the index of data element processed by the thread k in the source data array. The kc denotes index of the destination memory word in compressed data array, and *startbit* corresponds to the starting bit position inside that destination memory word.

Fig. 2 is given as an illustration of the parallel computation of the output index and starting bit position on a block of 8 input data elements: The first two steps of the parallel encoding algorithm result in the generation of matching codewords for the input symbols, codeword lengths (as the number of bits), and output parameters for the memory writes to the output data stream. The number of bits for each compressed data block is obtained as a byproduct of the first phase of the parallel prefix sum algorithm. Since a simple geometric decomposition is inherently applied on the GPUs as a step of the mapping process, this result can be used for concatenating the compressed data blocks into a contiguous array prior to data transfers from GPU to system memory.

3.3 Parallel Bit-Level Output

Bit-level I/O libraries designed for general-purpose CPUs process data serially, i.e., the codewords are stored one after the other into the memory. Implementation of a VLE on a parallel architecture introduces a problem of correctly dealing with race conditions that occur when different threads attempt to write their codewords to a same memory location. A recently introduced hardware support for atomic bitwise operations enables efficient execution of concurrent threads performing bit-level manipulations on the same memory locations, thus providing a mechanism for safely handling race conditions. The parallel output of codewords will produce correct results regardless of the write sequence, provided that each sequence of read-modify-write operations on a single memory location can be successfully completed without interruption, and that each output operation operation changes only the precomputed part of the destination word. The parallel bit-level output is executed in two stages: First, the contents

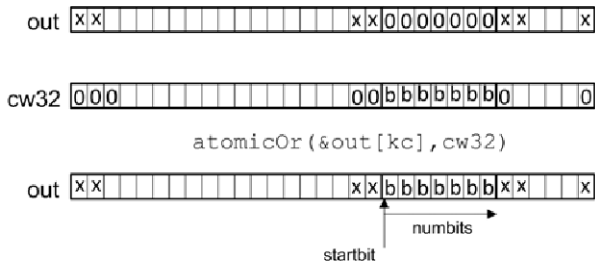


Fig. 3. Setting memory contents at index kc to the desired bit-values (codeword)

of the memory location at the destination address are prepared for the output by masking the *numbits* number of bits corresponding to the length of the codeword starting from the pre-computed bit output position. Second, the bits at these positions in the destination location are set to the value of the codeword, as illustrated in Fig. 3. If the contents of the destination memory are set in advance (all zeros), the output method can be reduced to only one atomic *or* operation. The implementation of the *put_bits_atomics* procedure for the GPGPUs supporting atomic operations (CUDA1.1+ compatible) is given in the code listing below.

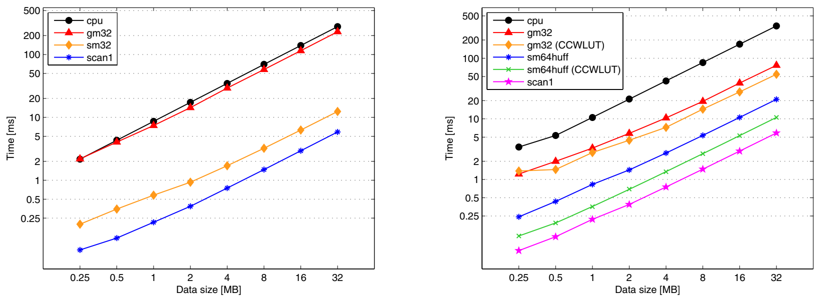
A situation when a codeword crosses boundaries of a destination word in memory can occur during variable length encoding, e.g., when the *startbit* is near the end of the current word, and the codeword to be written requires more bits than what is available in the reminder of the current word. The crossing of the word-boundary is detected and handled by splitting the output of the codeword into two or more store operations. When the codeword cross boundaries of several machine words, some of the atomic operations can be replaced by the standard store operation. The inner parts of the codeword can be simply stored to the destination memory location(s), and only the remaining bits on both sides of the codeword need to be set using the atomic operations.

```
__device__ void put_bits_atomic(unsigned int* out, unsigned int kc,
                    unsigned int startbit, unsigned int numbits,
                    unsigned int codeword) {

    unsigned int cw32 = codeword;
    unsigned int restbits = 32−startbit−numbits;

#ifndef MEMSET0
    unsigned int mask = ((1<<numbits)−1);
    mask <<= restbits;
    atomicAnd(&out[kc], ~mask);
#endif

    if ((startbit == 0) && (restbits == 0)) out[kc] = cw32;
        else atomicOr(&out[kc], cw32 << restbits);
}
```

4 Performance Results

Performance of several kernel implementations was benchmarked on a PC with an 2.66 GHz Intel QuadCore CPU, 2 GB RAM memory, and a NVIDIA GeForce GTX280 GPU supporting atomic instructions on 32-bit words. The test data set was composed of randomly generated test data files of different sizes and different amount of information content (entropy between 0.5-8 bits/symbol). The test files were assigned variable-length codewords using the Huffman algorithm with the restriction on the maximal codeword length. The performance of a CPU encoder running on one 2.66 GHz CPU core is given as a reference. Fig. 4(a) gives a performance comparison on a data set with 2.2 bits/symbol entropy. The GPU encoder *gm32* concatenates codewords for every 4 consecutive symbols (bytes) and writes the aggregate codeword to the GPU memory using global memory atomic operations. The performance of the serial encoder and the global memory (GM) encoder *gm32* are closely matching. However, by performing the atomic operations on a temporary buffer in shared memory (SM), as in *sm32*, a speed-up of more than an order of magnitude is achieved. The performance of the scan kernel, which is the asymptotically dominant part of the parallel algorithm, is given as a reference.

The *gm32* and *sm32* kernels operate under the assumption that the size of the aggregate codeword for four consecutive symbols (bytes) will not exceed the original data length, i.e. it will always fit into one 32-bit word. When using Huffman codewords, it may happen that the aggregate codeword exceeds the



(a) CPU and GPU Kernels

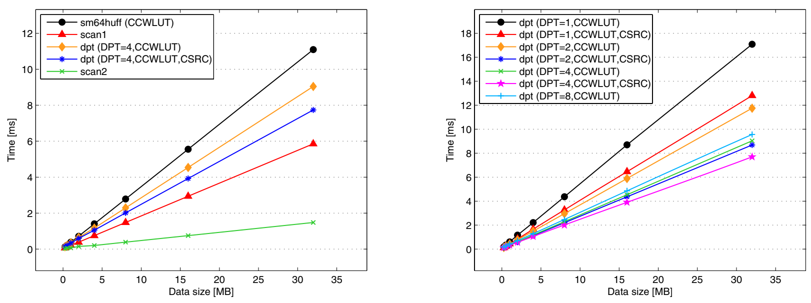
(b) Codeword LUT caching

Fig. 4. Kernel execution times as a function of data size

original data size. We designed a second SM kernel, denoted as *sm64huff*, that has a temporary buffer for the aggregate codeword of twice the original data size (a typical buffer size in compression implementations). The performance of *sm64huff* is slightly lower than the performance of *sm32* kernel, since it must perform one additional test during the codeword output. The situation when a codeword spans more than two destination memory locations is however correctly supported. In this case, no atomic operation is needed for the part of the codeword that spans an entire memory location, and a standard store operation can be used. However, empirical evaluation showed that atomic operations on the shared memory are implemented very efficiently, and that introduction of the additional test actually hurts the performance due to the increased warp serialization.

Additional performance improvements can be achieved by caching the codeword LUT, instead of looking up the codeword for each symbol in the global memory every time a symbol occurs. Fig. 4(b) gives a comparison of kernel execution times when the codeword look-ups are performed on the shared memory. Similar results are achieved by using the texture memory, which is cached by each multiprocessor. Use of low-latency shared memory for caching the codeword LUT improved the performance of GM kernels by approximately 20%, and the performance of SM kernels by up to 55%. As the symbols that appear more frequently are replaced by the codewords of shorter length, the low entropy data (well-compressible) will result in more shorter codewords that should be stored by different threads at the same memory location. This issue could be mitigated by processing more than one 32-bit data element per thread. The average number of bits that are written by each thread in one atomic operation to the destination memory location is increased and fewer atomic operations are issued.

Additionally, increasing DPT reduces the total number of data elements that is processed by the prefix sum (*scan*), which significantly influences the run time. Fig. 5(a) shows performance gains using the ideal DPT value; performance of *scan* using the original and reduced number of blocks are given as a reference.



(a) Comparison of standard and DPT kernels

(b) DPT Parameter effects

Fig. 5. Effects of processing more data per thread (lin scale)

Additional improvements are achieved by (1) caching the codeword LUT as previously described, and (2) caching aggregate codewords for every DPT elements in a local buffer. However, further increasing DPT radically increases memory requirements, since data is compressed in a shared memory buffer prior to transfer to the global memory. Fig. 5(b) gives a comparison of run times using several different DPT values. The investigation showed that the maximal DPT is limited by the shared memory requirements, and is relatively low ($DPT_{max} = 8$ when only codeword table is cached, and $DPT_{max} = 4$ when also aggregate codewords are cached). The best results are obtained using $DPT = 4$, resulting in a 35x speed-up.

5 Conclusion

In this paper, we presented a method for parallel bit-level output of data and a novel parallel algorithm for variable-length encoding (PAVLE) for GPGPU architectures supporting atomic operations. The PAVLE algorithm was implemented on a CUDA1.3-enabled GPGPU using atomic operations on the shared memory for managing concurrent codeword writes, parallel prefix sum for computing the codewords offsets in compressed data stream and caching of the codeword look-up tables in the low-latency memory. The optimized version of PAVLE for CUDA 1.3 compatible GPGPUs achieves performance of approximately 4GB/sec using Huffman codes for encoding the data on the NVIDIA GeForce GTX280 GPGPU. We observed considerable speedups compared to the serial VLE on the state of the art PCs (up to 35x on 2.66GHz CPU, and up to 50x on a 2.40GHz CPU), thus making the PAVLE an attractive lossless compression algorithmic building block for GPGPU-based applications.

Acknowledgments. The authors would like to thank Glenn R. Luecke for his support and his invaluable comments on the manuscript. Special thanks go to Tjark Bringewat for benchmarking and very constructive discussions on GPGPU kernel optimizations.

References

1. Huffman, D.: A method for the construction of Minimum-Redundancy codes. Proceedings of the IRE 40(9), 1098–1101 (1952)
2. Allusse, Y., Horain, P., Agarwal, A., Saipriyadarshan, C.: GpuCV: an opensource GPU-accelerated framework forimage processing and computer vision. In: 2006 IEEE International Conference on Multimedia and Expo. (2008)
3. Chen, W., Hang, H.: H. 264/AVC motion estimation implmentation on Compute Unified Device Architecture (CUDA). In: 2008 IEEE International Conference on Multimedia and Expo., pp. 697–700 (2008)
4. Fung, J., Mann, S.: Using graphics devices in reverse: GPU-based image processing and computer vision. In: 2008 IEEE International Conference on Multimedia and Expo., pp. 9–12 (2008)
5. Blelloch, G.E.: Prefix sums and their applications. Synthesis of Parallel Algorithms, 35–60 (1990)
6. Roger, D., Assarsson, U., Holzschuch, N.: Efficient stream reduction on the gpu. In: Kaeli, D., Leeser, M. (eds.) Workshop on General Purpose Processing on Graphics Processing Units (October 2007)
7. Castaño, I.: High quality dxt compression using cuda. Technical report, NVIDIA (last access, May 2008)
8. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro 28(2), 39–55 (2008)
9. NVIDIA Corporation Technical Staff: Nvidia cuda -programming guide 2.0. Technical report, NVIDIA (last access, May 2009)
10. Atallah, M., Kosaraju, S., Larmore, L., Miller, G., Teng, S.: Constructing trees in parallel. In: Proceedings of the first annual ACM symposium on Parallel algorithms and architectures, pp. 421–431. ACM, New York (1989)
11. Harris, M., Sengupta, S., Owens, J.D.: Parallel prefix sum (scan) with cuda. GPU Gems 3 (2007)

Towards Metaprogramming for Parallel Systems on a Chip*

Lee Howes[1], Anton Lokhmotov[1], Alastair F. Donaldson[2], and Paul H.J. Kelly[1]

[1] Department of Computing, Imperial College London, 180 Queen's Gate, London, SW7 2AZ, UK
[2] Computing Laboratory, University of Oxford, Parks Road, Oxford, OX1 3QD, UK

Abstract. We demonstrate that the performance of commodity parallel systems significantly depends on low-level details, such as storage layout and iteration space mapping, which motivates the need for tools and techniques that separate a high-level algorithm description from low-level mapping and tuning. We propose to build a tool based on the concept of decoupled Access/Execute metadata which allow the programmer to specify both execution constraints and memory access pattern of a computation kernel.

1 Introduction

We evaluate several implementations of simple image filters on x86 multicore systems and a GPU-accelerated system. Our experimental results demonstrate that efficiently implementing an algorithm to execute on commodity parallel hardware requires careful tuning to match the hardware characteristics, as the performance depends significantly on low-level details such as iteration space mapping and storage layout. While such manual tuning is possible, it is not practical: the number of versions to write and maintain grows with the number of target architectures. For applications consisting of multiple kernels such development and maintenance becomes infeasible.

We believe that innovative tools and techniques that separate a high-level algorithm description from low-level mapping and tuning will make software engineering for parallel systems more productive and disciplined. We propose to build such a tool based on the concept of Access/Execute (Æcute) metadata which capture both execution constraints and memory access patterns [1].

2 Mean Filter

Consider a one-dimensional mean filter, for which the output at t is given by the formula

$$\mathbf{O}_t = \frac{1}{D}\sum_{k=0}^{D-1}\mathbf{I}_{t+k}, \text{ where} \tag{1}$$

* We acknowledge financial support by the EPSRC grants EP/E002412 and EP/G051100.

- **I** is an input array of $N + D$ real elements;
- **O** is an output array of N real elements;
- D is the *diameter* of the filter, *i.e.* the number of input elements over which the mean is computed (typically, $D \ll N$).

A naïve parallel algorithm can run N threads, each producing a single output element, which requires $\Theta(ND)$ reads and arithmetic operations. A good parallel algorithm, however, must be efficient and scalable [2].

2.1 Scalable Algorithm

The algorithm in Listing 1 *strips* the computation, where up to T outputs in the same strip are computed serially in two *phases*. The first phase in lines 2–6 computes $\mathbf{O}_{t0}$ according to (1). The second phase in lines 8–14 computes $\mathbf{O}_t$ for $t \geq t0 + 1$ as $\mathbf{O}_{t-1} + \big(\mathbf{I}_{t+D-1} - \mathbf{I}_{t-1}\big)/D$.

```
for(int t0 = 0; t0 < N; t0 += T) {                          1
    // first phase: convolution                             2
    float sum = 0.0f;                                       3
    for(int k = 0; k < D; ++k)                              4
        sum += I[t0+k];                                     5
    O[t0] = sum / (float)D;                                 6
                                                            7
    // second phase: rolling sum                            8
    for(int dt = 1; dt < min(T,N-t0); ++dt) {               9
        int t = t0 + dt;                                   10
        sum -= I[t-1];                                     11
        sum += I[t-1+D];                                   12
        O[t] = sum / (float)D;                             13
    }                                                      14
}                                                          15
```

Listing 1. Scalable mean filter algorithm in C

This algorithm performs $\Theta(N + ND/T)$ reads and arithmetic operations, considerably reducing memory bandwidth and compute requirements for $T \gg D$. Since the $t0$ loop carries no dependences, up to $\lceil N/T \rceil$ threads can run in parallel. Thus, this algorithm trades off parallelism against work efficienty.

Note that since the order of arithmetic operations is undefined in (1), both the naïve and scalable algorithms are functionally, if not arithmetically, equivalent.

2.2 Vertical and Horizontal Mean Image Filters

Mean filtering is a simple technique for reducing noise in digital images.

The vertical mean image filter is the one-dimensional mean filter applied to columns of a two-dimensional image of $W \times H$ pixels:

$$\mathbf{O}_{x,y} = \frac{1}{D}\sum_{k=0}^{D-1} \mathbf{I}_{x,y+k}, \text{ where } 0 \leq x < W, 0 \leq y < H - D. \tag{2}$$

Similarly, the horizontal mean image filter is the mean filter applied to rows of an image:

$$\mathbf{O}_{x,y} = \frac{1}{D}\sum_{k=0}^{D-1} \mathbf{I}_{x+k,y}, \text{ where } 0 \leq x < W - D, 0 \leq y < H. \tag{3}$$

Using the algorithm of §2.1, the mean image filters perform $\Theta(N + ND/T)$ reads and arithmetic operations, and can run up to $\lceil N/T \rceil$ parallel threads, where N is the number of output pixels and T is the number of output pixels per strip.

Clearly, the optimal value of T depends on problem parameters W, H and D, and on hardware parameters such as the number of supported threads and memory bandwidth. In our evaluation (§4), we find the optimum by iteratively compiling the kernels for a range of values of T and evaluating the performance.

3 Implementation

We describe efficient implementations of the mean image filter kernels for a GPU, with the NVIDIA Compute Unified Device Architecture (CUDA) [3], and for a multicore CPU, with Intel Streaming SIMD Extensions (SSE) [4]. We assume that the image pixels are represented as single-precision floating-point numbers, and that the images are stored in row-major order.

3.1 Architecture Overview

Roughly, a CUDA thread corresponds to an individual SSE vector lane, whilst a CUDA thread block corresponds to a full SSE vector. A GPU core (streaming multiprocessor) executes blocks of multiple threads in SIMD groups of 32 threads (warps) using the 8-lane SIMD unit; a CPU core operates on 4-element vectors using the 4-lane SIMD unit. As a rule of thumb, a GPU runs thousands of threads, whilst a CPU only tens of threads (counting vector lanes).

SSE only supports *coalesced* access to off-chip memory, *e.g.* storing a vector register into a contiguous (and preferably aligned) 128-bit memory region; CUDA supports uncoalesced access to off-chip memory, albeit at a lower memory bandwidth.[1] To reduce off-chip memory access, SSE provides a family of instructions for shuffling data in vector registers, whilst CUDA provides on-chip memory shared between threads in a block. Effectively, these mechanisms enable fast inter-thread cooperation.

3.2 Vectorisation

Vertical mean filter. Conceptually, the vertical mean filter has a parallel outer loop iterating over each column and a parallel inner loop iterating over strips of rows:

[1] Rules for coalescing vary between different architecture generations.

```
parfor(x = 0; x < W; ++x) // for each column
  parfor(y0 = 0; y0 < H-D; y0 += T) // for each strip of rows
    // two-phase computation here
```

To enable memory coalescing, threads in a thread block (lanes in a vector) must access a contiguous (and preferably aligned) memory region, which is achieved by assigning adjacent columns to adjacent threads: technically, the loop x is interchanged with the loop $y0$, and then stripmined into vectors.

Horizontal mean filter. The horizontal mean filter has a parallel outer loop iterating over each row and a parallel inner loop iterating over strips of columns:

```
parfor(y = 0; y < H; ++y) // for each row
  parfor(x0 = 0; x0 < W-D; x0 += T) // for each strip of columns
    // two-phase computation here
```

Serialisation within strips of columns, however, results in *no* memory coalescing: adjacent threads access adjacent rows with a stride of W. One option, illustrated in Fig. 1, is to transpose the $W \times H$ input image $\mathbf{I}$ to an $H \times W$ intermediate image $\mathbf{T}'$, run the vertical mean filter on $\mathbf{T}'$ to produce an $H \times (W - D)$ intermediate image $\mathbf{T}''$, and then transpose $\mathbf{T}''$ to produce the $(W - D) \times H$ output image $\mathbf{O}$. Another option is to effectively *fuse* transposition and computation into one optimised kernel, by using on-chip memory on a GPU or shuffle instructions on a CPU.

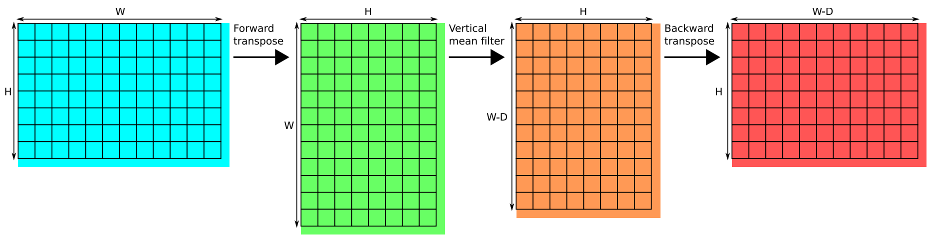
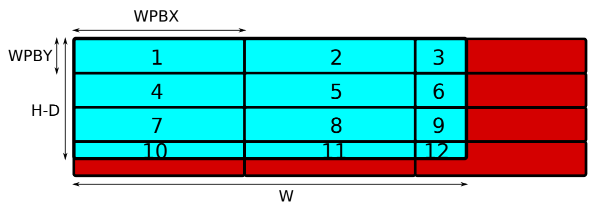


Fig. 1. The horizontal mean filter kernel implemented as a pipeline of the forward transpose, vertical mean filter and the backward transpose kernels. Shadows represent the padding that may be necessary to improve the bandwidth of the transpose kernels (§4.1).

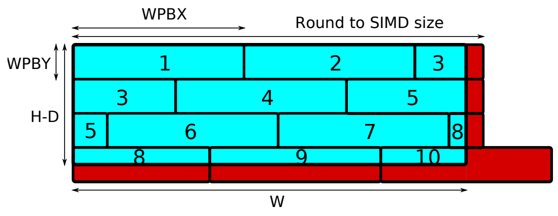
3.3 Parallelisation

Given a low thread count on a CPU, the inner loop computation can be completely serialised, resulting in maximum work efficiency: the CPU out-of-order issue logic can extract adequate instruction level parallelism from the serial instruction stream. In contrast, a GPU exploits limited instruction level parallelism and relies on thread level parallelism to cover memory latency.

On a GPU, thread blocks are located in a grid. For two-dimensional iteration spaces over images, a two-dimensional grid is most natural, with each block producing a rectangular section of the output image. As Fig. 2a illustrates for the vertical mean filter, significant portions of thread blocks covering the right edge of the image may be idle if the image width is not a multiple of the number of columns per thread block.



(a) A 2D grid mapping loses efficiency from idle threads off the right image edge



(b) A 1D grid mapping has fewer idle threads by wrapping around the right image edge. Taking into account alignment for efficiency reasons complicates addressing and iteration.

Fig. 2. Different mapping strategies result in different utilisation of threads. Light and dark regions of blocks denote busy and idle threads, respectively. WPBX and WPBY stand for *work per block* in the x and y dimensions, respectively. For 128 $\times$ 1 thread blocks used in our evaluation (§4), WPBX = 128 and WPBY = T.

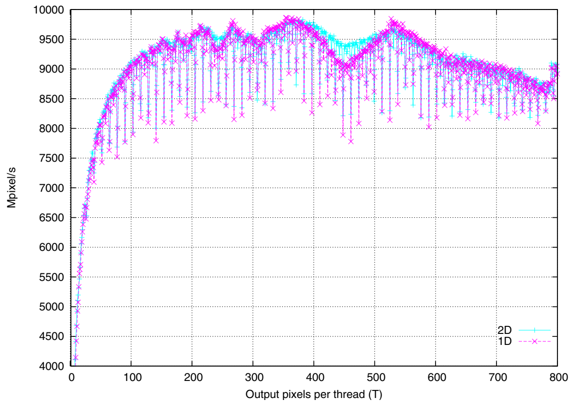
This issue can be alleviated by mapping the iteration space onto a one-dimensional grid that covers the image by wrapping around its right edge, as illustrated by Fig. 2b. As we show in §4.1, a mapping that maximises thread utilisation suffers from misalignment, if the image width is not a multiple of the warp size; a better mapping takes alignment into account by wasting a small number of threads on the right of the image, thus ensuring that the first pixel of each row is handled by the first thread in a warp.

4 Experimental Results

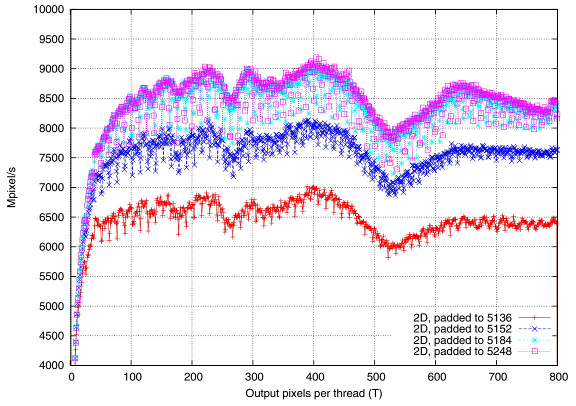
4.1 GPU

We present results obtained on a dual-core 3GHz Intel Core 2 Duo E8400 system with 2 GiB RAM, equipped with an NVIDIA GTX 280 card, running 64-bit Linux Ubuntu 8.04. Code is compiled using CUDA SDK 2.2 and GCC 4.2.4 with the "-O3" optimisation setting. We measure the kernel execution time only and report the best throughput out of 50 runs. In all the experiments, we fix the number of threads per block at 128 (128 $\times$ 1), as we nearly achieve the peak memory efficiency with this setting: $\approx$ 10 Gpixel/s $\times$ 4 bytes/pixel $\times$ (2 reads + 1 write) = 120 GB/s (close to the bandwidth of aligned copy on this card).

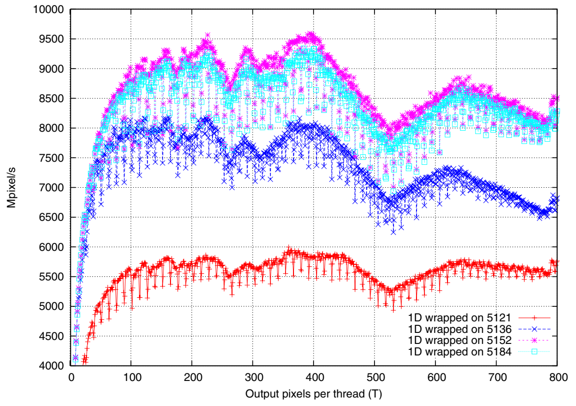
Vertical mean filter. Fig. 3a shows that the 1D and 2D grid versions of the vertical mean filter (Fig. 2) are similar in throughput when applied to a 5120 $\times$ 3200 image, where 5120 is a multiple of 128 pixels. The throughput is below 0.8 Gpixel/s (not



(a) 5120 × 3200 image. 2D grid; 1D grid.



(b) 5121 × 3200 image. 2D grid. Data padded to multiples of 16, 32, 64, and 128 pixels.



(c) 5121 × 3200 image. Data padded to 5184 (a multiple of 64) pixels. 1D grid wrapped on the image width and multiples of 16, 32 and 64 pixels.

Fig. 3. CUDA implementations for the vertical mean filter on a 5120 × 3200 image (a) and on a 5121 × 3200 image (b & c), with different iteration space mapping and storage layout

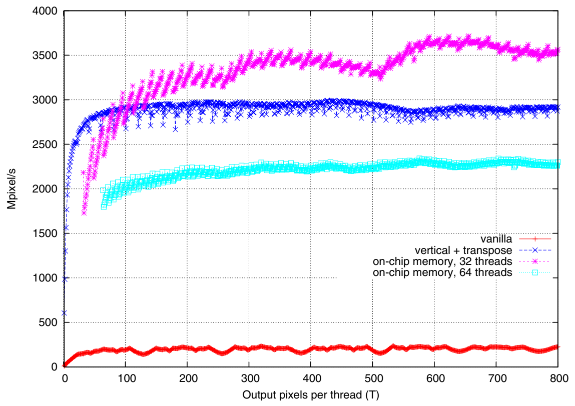


Fig. 4. CUDA implementations for the horizontal mean filter on a 5120×3200 image

shown) when each thread produces a single pixel ($T = 1$), climbs fast with increasing serial efficiency, achieving the peak throughput of 9.9 Gpixel/s at several points, and then declines with decreasing parallelism.

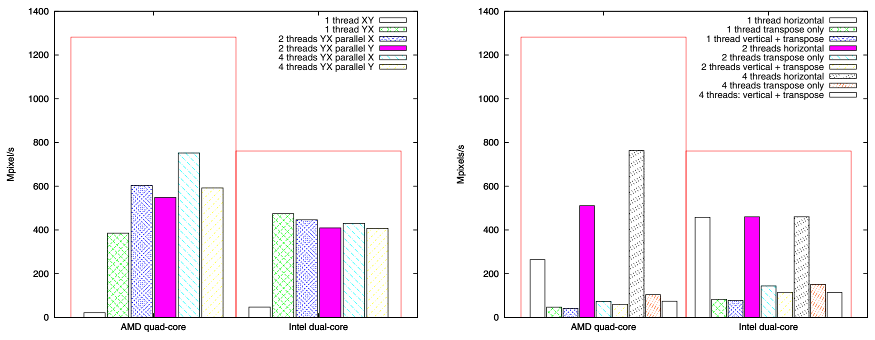
When applied to a 5121×3200 image, however, the 2D grid version only achieves 7.0 Gpixel/s, as shown by the bottom line in Fig. 3b. Whilst we allocate memory using the `cudaMallocPitch` function, which pads the image to a multiple of 16 pixels to enable memory coalescing (5136 pixels in this case), such allocation leads to DRAM partition conflicts. We remedy the conflicts by manually padding the image to a multiple of 32, 64 and 128. Since the results of padding to a multiple of 64 and 128 are very close, we fix the image padding at a multiple of 64 (5184 pixels) for all subsequent experiments.

Fig. 3c shows that the 1D grid mapping that maximises thread utilisation by wrapping on 5121 pixels hardly achieves 6.0 Gpixel/s, whilst wrapping on the image padding of 5184 pixels performs worse than wrapping on the warp size multiple of 5152 pixels.

To summarise, for the misaligned image padded to 5184 pixels, the 1D grid version wrapped on 5152 pixels achieves 9.6 Gpixel/s, whilst the 2D grid version achieves only 9.1 Gpixel/s; thus, the 1D grid version is 6% faster than the 2D grid version.

Horizontal mean filter. Fig. 4 shows that the vanilla horizontal mean filter version on a 5120×3200 image achieves only 230 Mpixel/s, for most values of T. The version that uses on-chip memory to effectively fuse transposition and computation into one optimised kernel, achieves 2.4 Gpixel/s and 3.7 Gpixel/s when using 64 and 32 threads.

The composite version that uses separate transpose and vertical mean kernels (Fig. 1) achieves 3.0 Gpixel/s. Note, however, that this performance is only achieved when the intermediate images $\mathbf{T}'$ and $\mathbf{T}''$ are both padded by 64 pixels, resulting in the bandwidth of 80.1 GiB/s and 63.3 GiB/s for the forward and backward transposes, respectively. Without the padding, the bandwidth is only 60.2 GiB/s and 19.1 GiB/s.



(a) The vertical mean filter (b) The horizontal mean filter

Fig. 5. Comparison of different CPU implementations on a 5120×3200 image. The large surrounding boxes represent the peak memory copy throughput for each of the systems, as obtained by running the STREAM benchmark [5].

We estimate that assembling the composite version from the already available components took half a day versus five days for the on-chip memory version. Tuning the kernels and finding the optimal padding parameters to improve the bandwidth, however, took another half a day.

4.2 CPU

We present results on a 2.3 GHz quad-core AMD Phenom 9650 system with 8 GiB RAM (AMD) and on a 3 GHz dual-core Intel Core 2 Duo E8400 system with 2 GiB RAM (Intel), both running 64-bit Linux Ubuntu 8.04. Code is parallelised using OpenMP pragmas and compiled using Intel C Compiler 11.0 with the “-xHost -fast” setting.

Vertical mean filter. In the worst performing version in Fig. 5a (“XY”), the loop over columns x and the loop over strips of rows $y0$ have not been interchanged, which results in strided memory accesses. Applying loop interchange (“YX”) results in a vast performance increase. Performance on the AMD system increases with enabling more cores, whilst the Intel system achieves the peak performance with a single core, which can be attributed either to the compiler better optimising for Intel’s own architecture or to lower performance of a single core on the AMD system. On both systems, it is always more beneficial to parallelise across multiple cores the x loop (“parallel X”) than the $y0$ loop (“parallel Y”).

Horizontal mean filter. The best performing version in Fig. 5b is obtained by the Intel compiler optimising a naïve C implementation that runs through memory sequentially in the horizontal dimension (“horizontal”), thus triggering the CPU cache prefetching mechanism. On the CPUs, the forward and backward transposes are too costly even when using the highly optimised Intel MKL library (“transpose only”), and adding a best performing version of the vertical mean filter (“vertical + transpose”) makes little difference. Indeed, the naïve implementation is so fast that there is nothing to gain from vectorisation but a lot to lose from transposition.

5 Towards Metaprogramming

5.1 Decoupled Access/Execute Metadata

To ease the programmer's burden of mapping and tuning computation kernels to parallel systems on a chip, we propose extending a kernel's description with decoupled Access/Execute (Æcute) metadata [1]. *Execute* metadata for a kernel describe its iteration space ordering and partitioning; *access* metadata describe locations in uniform memory that the kernel may access on each iteration.

```
// Array descriptors (C array wrappers)                     1
Array2D<float> arrayI(&I[0][0], W, H);                      2
Array2D<float> arrayO(&O[0][0], W, H-D);                    3
                                                            4
// Execute metadata: parallel iteration space               5
IterationSpace1D x(0,W);                                    6
IterationSpace1D y(0,H-D);                                  7
IterationSpace2D iterXY(x,y);                               8
                                                            9
// Access metadata: iteration space -> memory               10
VerticalStrip2D_R accessI(iterXY, arrayI, D);               11
Point2D_W accessO(iterXY, arrayO);                          12
```

Listing 2. Æcute metadata for the vertical mean image filter

Listing 2 gives an example of Æcute metadata for the vertical mean image filter. Accesses to plain C arrays `I[W][H]` and `O[W][H-D]` are wrapped using Æcute array descriptors `arrayI` and `arrayO` to cleanse the kernel of uncontrolled side-effects (lines 1–3). A 2D iteration space descriptor `iterXY` is constructed from 1D descriptors `x` and `y`, having the same bounds as the output image dimensions (lines 5–8). By default, an iteration space is parallel in every dimension. Finally, we specify that on each iteration the kernel reads a vertical strip of D pixels from `arrayI` and writes a single pixel to `arrayO` (lines 10–12).

5.2 Related Work and Discussion

Effectively orchestrating data movement in software-managed memory hierarchies is paramount to achieving high performance but is tedious and error-prone. The CUDA-lite [6] tool seeks to automate data movement between on-off chip and on-chip GPU memories by generating appropriate code from ad-hoc source code annotations. We have addressed the problem of generating data movement code between two levels of memory hierarchy on the Cell BE architecture in our previous work [1]. We now aim to address a broader problem of generating code for both data movement across the full memory hierarchy (including the host memory) *and* the iteration space traversal.

Similar to the Sequoia language [7], we seek to separate a high-level algorithm representation from a system-specific mapping. Unlike Sequoia, we base our mapping on partitioning (manually or automatically) an iteration space into disjoint subspaces and infer memory access of subspaces from Æcute metadata. For a GPU-accelerated system, a hierarchy of iteration space partitions can specify subspaces to be executed: at the lowest level, by individual threads; at the middle level, by blocks of possibly cooperating threads; at the highest level, by possibly cooperating compute devices:

```
iterXY.partitionThreads(1,T);      // 1xT outputs/thread
iterXY.partitionBlocks(128,T);     // 128xT outputs/block
iterXY.partitionDevices(W/2,H-D); // (W/2)x(H-D) outputs/device
```

Ryoo *et al.* [8] also highlight the need for design space exploration, which they call *optimization carving*, but leave out the question of automatically generating different code versions from a high-level representation.

6 Future Work

OpenCL [9], a new low-level standard API, aims to provide software portability across heterogeneous systems. Thus, instead of writing, for example, separate CUDA and SSE kernels, the programmer will be able to write an OpenCL kernel and run it on any standard-compliant implementation. However, OpenCL per se does not address the problem of *performance portability*, since low-level code optimised for one device may perform dismally on another, as we have demonstrated in this paper.

We aim to tackle this problem by designing and implementing a framework that will take a device-independent algorithm representation with Æcute metadata and generate efficient device-specific OpenCL code to be processed by vendor compilers.

References

1. Howes, L.W., Lokhmotov, A., Donaldson, A.F., Kelly, P.H.: Deriving efficient data movement from decoupled Access/Execute specifications. In: Seznec, A., Emer, J., O'Boyle, M., Martonosi, M., Ungerer, T. (eds.) HiPEAC 2009. LNCS, vol. 5409, pp. 168–182. Springer, Heidelberg (2009)
2. Lin, C., Snyder, L.: Principles of Parallel Programming. Addison-Wesley, MA (2008)
3. NVIDIA: CUDA (2006–2009), `http://www.nvidia.com/cuda`
4. Bik, A.J.: The Software Vectorization Handbook. Applying Multimedia Extensions for Maximum Performance. Intel Press, Hillsboro (2004)
5. McCalpin, J.D.: STREAM: Sustainable memory bandwidth in high performance computers (1990–2009), `http://www.cs.virginia.edu/stream/`
6. Ueng, S.-Z., Lathara, M., Baghsorkhi, S.S., Hwu, W.m.W.: CUDA-lite: Reducing GPU programming complexity. In: Amaral, J.N. (ed.) LCPC 2008. LNCS, vol. 5335, pp. 1–15. Springer, Heidelberg (2008)
7. Fatahalian, K., Horn, D.R., Knight, T.J., Leem, L., Houston, M., Park, J.Y., Erez, M., Ren, M., Aiken, A., Dally, W.J., Hanrahan, P.: Sequoia: programming the memory hierarchy. In: Supercomputing'06, p. 83 (2006)
8. Ryoo, S., Rodrigues, C.I., Stone, S.S., Stratton, J.A., Ueng, S.Z., Baghsorkhi, S.S., Hwu, W.m.W.: Program optimization carving for GPU computing. J. Parallel Distrib. Comput. 68(10), 1389–1401 (2008)
9. The Khronos Group: OpenCL (2008–2009), `http://www.khronos.org/opencl`

Dynamic Detection of Uniform and Affine Vectors in GPGPU Computations

Caroline Collange[1], David Defour[1], and Yao Zhang[2]

[1] ELIAUS, Université de Perpignan, 66860 Perpignan, France
{caroline.collange@inria.fr,david.defour}@univ-perp.fr
[2] ECE Department, University of California Davis
yaozhang@ucdavis.edu

Abstract. We present a hardware mechanism which dynamically detects uniform and affine vectors used in SPMD architecture such as Graphics Processing Units, to minimize pressure on the register file and reduce power consumption with minimal architectural modifications. A preliminary experimental analysis conducted with the Barra simulator shows that this optimization can benefit up to 34 % of register file reads and 22 % of the computations in common GPGPU applications.

1 Introduction

GPUs are now powerful and programmable processors that have been used to accelerate general-purpose tasks other than graphics applications. These processors rely on a Single Program Multiple Data (SPMD) programming model. This model is implemented by many vector units working in a Single-Instruction Multiple-Data (SIMD) fashion, and vector register files. Register usage is a critical issue as the number of instance of the same program that can be executed simultaneously depend on the number of hardware registers and the register usage per instance. Making this bad situation worse, vectorizing scalar operations in an application makes inefficient use of registers and functional units. To efficiently handle scalar data, Cray-like vector machines incorporate scalar functional units as well as scalar registers. Modern GPUs lack such scalar support, leaving it to vector units. These vector units execute the same instruction on the same data leading to as many unnecessary operations as the length of the vector when uniform data are encountered. These unnecessary operations involve data transfers and activity in functional units that consume power, which is a critical concern in architectural and microarchitectural designs of GPUs.

We observed through our experiments that standard GPGPU applications manipulate a significant number of degenerated vectors containing replicated scalar data, that we name *uniform* vectors. A closer look shows that this number is even higher when additionally considering *affine* vectors, which contain a sequence of regularly-stepped values (e.g. $(2, 4, 6 \ldots, 2n + 2)$) Motivated by this observation, we propose and evaluate by simulation a technique that tags a vector register file according to the type of registers: uniform, affine or *generic* (non-degenerate) vector.

The rest of the paper begins with a brief description of the NVIDIA architecture upon which our model is based. Section 3 presents our performance evaluation methodology, based on a functional simulator named Barra. We use it to both evidence the presence of redundancy in calculations in Section 4 and evaluate the proposed technique described in Section 5. We discuss technical issues in Section 6. Section 7 presents quantitative results and figures, and Section 8 concludes the paper.

2 Architecture Model

The base architecture we consider in our simulations consists of a vector processor, a set of vector register files, a set of vector units, and an instruction set architecture that mimics the behavior of the NVIDIA GPUs used in the Compute Unified Device Architecture (CUDA) environment [1]. This environment relies on a stack composed of an architecture, a language, a compiler, a driver and various tools and libraries.

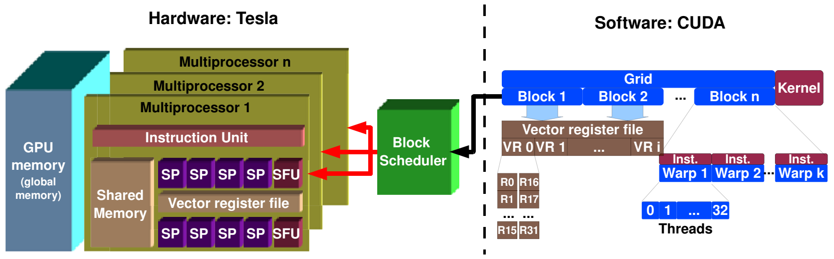


Fig. 1. Processing flow of a CUDA program

A CUDA program runs on an architecture composed of a host processor CPU, a host memory and a graphics card with an NVIDIA GPU with CUDA support. All current CUDA-enabled GPUs are based on the Tesla architecture, which is made of an array of *multiprocessors*. Tesla GPUs execute thousands of threads in parallel thanks to the combined use of multiple multiprocessors, SIMD processing and hardware multithreading [2]. Figure 1 describes the hardware organization of such a processor. Each multiprocessor contains the logic to fetch, decode and execute SIMD instructions which operate on vectors of 32 elements. There are 256 or 512 vector registers, each register being a 32-wide vector of 32-bit values. In addition to the register file, each multiprocessor contains a scratchpad memory (or shared memory, using NVIDIA's terminology) and separate caches for constant data and instructions.

The hardware organization is tightly coupled with the parallel programming model of CUDA. The programming language used in CUDA is based on C with extensions to indicate if a function is executed on the CPU or the GPU. Functions executed on the GPU are called *kernels*. CUDA lets the programmer define if a variable resides in the GPU address space and specify the kernel execution across different granularities of parallelism: *grids*, *blocks* and *threads*. As the underlying hardware is a SIMD processor, threads are grouped together in so-called *warps* which operate on 32-wide vector registers. Each instruction is executed on a warp by a multiprocessor. Warps execute instructions at their own pace, and multiple warps can run concurrently on a multiprocessor to hide latencies of memory and arithmetic instructions. This technique helps hide the latency of streaming transfers, and improve the effective memory bandwidth. The register file of a multiprocessor is logically split between the warps it executes. As a GPU includes several multiprocessors, warps are grouped into blocks. Blocks are scheduled on the available multiprocessors. A multiprocessor can process several blocks simultaneously if enough hardware resources (registers and shared memory) are available.

The compilation flow of a normal CUDA program is a three-step process directed by the CUDA compiler *nvcc*. First, according to specific CUDA directives from the CUDA Runtime API, the program is split into a host program and a device program. The host program is then compiled using a host C or C++ compiler and the device program is compiled through a specific back-end for the GPU. The resulting device code is binary instruction code (in *cubin* format) to be executed on a specific GPU. The host program and the device program are linked together using the CUDA libraries, which includes the necessary functions to load a cubin either from inside the executable or from a stand-alone file and send it to the GPU for execution.

3 Barra, a Functional Simulator of NVIDIA GPUs

Several options exist to model the dynamic behavior of CUDA programs. CUDA offers a built-in emulation mode that run POSIX threads on the CPU on behalf of GPU threads, thanks to a specific compiler back-end. However, this mode differs in many ways with the execution on a GPU: the behavior of floating-point and integer computations, the scheduling policies and memory organization are different.

GPU simulators running CUDA's intermediate language PTX such as GPGPU-Sim [3] or Ocelot [4] can offer a greater accuracy, but still run an unoptimized intermediate code instead of the instructions actually executed by a GPU.

Recent versions of CUDA include a debugger that allows watching the values of GPU registers between each line of source code. Though this mode offers perfect functional accuracy, it cannot be modified for instrumentation or feature evaluation purposes.

Barra [5] simulates the actual instruction set of the NVIDIA Tesla architecture at the functional level. The behavior of all instructions is reproduced with bit-accuracy, with the exception of transcendentals (exp, log, sin, cos, rcp, rsq). To our knowledge, Barra is the only publicly-available tool that both executes the same instructions as Tesla GPUs and allows viewing the exact contents of registers during the execution.

This simulator consists of two parts: a driver, and a simulator. The driver is a shared library with the same name and exporting the same symbols as NVIDIA's *libcuda.so* so that CUDA Driver API calls can be dynamically redirected to the simulator. It includes major API functions to load, and execute a CUDA program and manage data transfers. The simulator takes the binary code compiled by NVIDIA's nvcc compiler as input simulates the execution of the kernel, and produces statistics for the each instruction.

3.1 Logical Execution Pipeline

The instruction-set simulator executes each assembly instruction according to the model described in Figure 2. First, a scheduler selects the warp ready for execution according to a round-robin policy and reads its current program counter (PC). Then the instruction is fetched and decoded. Then operands are read from the register file or from on-chip memories (scratchpad) or caches (constants). The instruction is executed and its results are written back to the register file.

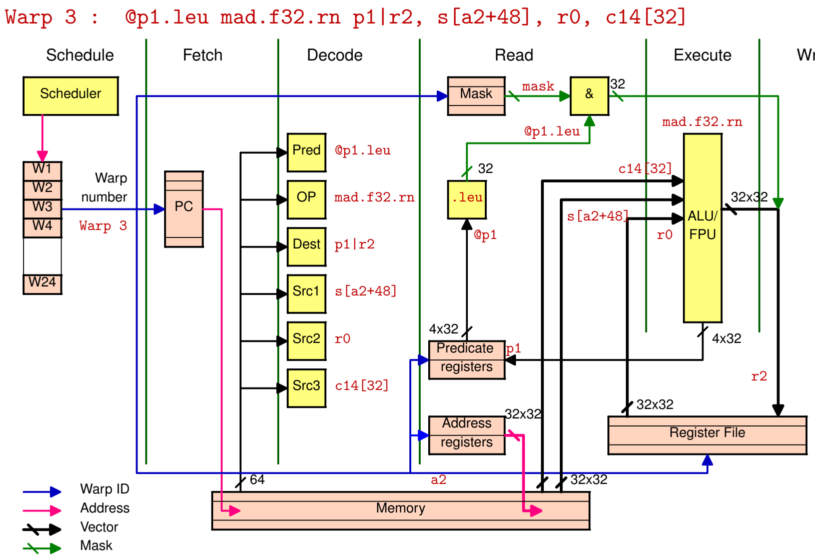


Fig. 2. Overview of the functional execution pipeline during the execution of a MAD instruction

3.2 Vector Register File

General Purpose Registers (GPRs) are dynamically split between threads during kernel launch, allowing a trade-off between the number of registers per threads and the latency hiding capability. Barra maintains a separate state for each active warp in the multiprocessor. These states include a program counter, address and predicate registers, mask and address stacks, a window to the assigned register set, and a window to the shared memory.

4 Uniform and Affine Data in SPMD Code

NVIDIA's so-called "scalar" architecture is actually a pure vector (SIMD) architecture. At the hardware level, all instructions, including memory and control-flow operations, operate on vectors, and all architecturally-visible registers are vectors. It can alternatively be seen from the programer's perspective as a SPMD machine executing independent "scalar" threads. Though this provides a developer-friendly programming model and allows scalable implementations by abstracting away the vector length, it can become a source of inefficiency when performing inherently scalar operations.

We define an *uniform vector* V as having every component contain the same value $V_i = x$. Two main causes lead to uniform vector patterns. First, constant values and data read from memory with a uniform address vector (broadcast) generate uniform vectors. Second, uniform control flow is governed by uniform conditions. For example, a *for* loop with uniform bounds will also have a uniform counter. Modern GPUs also allow non-uniform conditions in conditional statements by allowing sub-vector control-flow. However, best performance is achieved when the control condition is uniform across a warp [1]. This means that in optimized algorithms, all lanes of registers that are used as conditions will hold the same value.

Similarly, to maximize memory bandwidth, memory accesses should follow specific patterns, such as the coalescing rules or conflict-free shared-memory access rules. Programs following NVIDIA's guidelines to access memory will operate mostly on consecutive addresses. This specific pattern is often referred as *unit-stride access* in the vector-computing literature. The vector register storing the addresses of such access will contain consecutive addresses.

This leads us to define an *affine* vector as a vector V having each of its component (interpreted as an unsigned integer) such that $V_i = x + iy$, for x and y non-negative integers. One can notice that the uniform pattern is a specific case of the affine pattern when $y = 0$.

To quantify how often both of these patterns occur, we use Barra to dynamically check for each input and output operand in registers if they are uniform or affine vectors. We perform this analysis on two kinds of applications.

First, we used the examples from the CUDA SDK. Even though these examples are not initially meant to be used as benchmarks, they are currently the most standardized test suite of CUDA applications. As code examples, they reflect the best practices in CUDA programming.

The second benchmark is a bioinformatics application. RNAFold_GPU is a CUDA program which performs RNA folding. Based on dynamic programming, it achieves a 17-time speedup compared to a multicore implementation [6].

The proportion of uniform and affine inputs/outputs from and to registers is depicted in figure 3. Uniform or affine input data represent the percentage of uniform (affine) vectors among the data transferred between the register file and functional units.

Similarly, uniform or affine output data is the proportion of uniform (affine) data written back to the register file. It can be observed that whenever the output is uniform (affine), the operation itself is executed on uniform (affine) data only.

We observe that a respective average of 27 % (44 %) of data read from the vector register file are uniform (affine) and 15 % (28 %) of data written back are uniform (affine). This proportion of uniform or affine inputs/outputs is significant enough to justify specific optimizations.

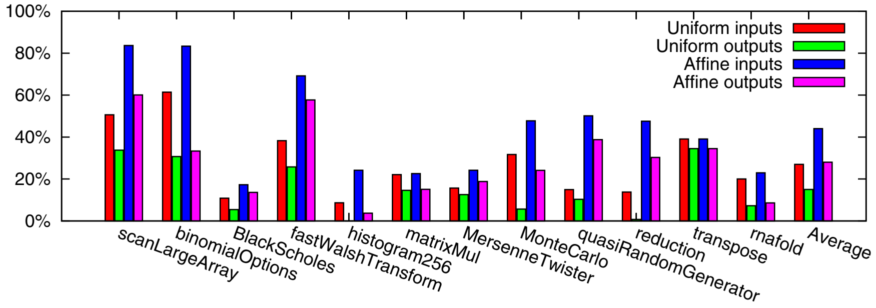


Fig. 3. Proportion of uniform and affine operands in registers. Averages are 27 % uniform inputs, 15 % uniform outputs, 44 % affine inputs and 28 % affine outputs.

5 Proposed Technique

In this section we describe a technique which can detect if registers contain uniform or affine data as defined in Section 4. The first objective is to minimize memory and bus activity between the register file and the functional units for the proportion of input data captured by the proposed technique. The second objective extends the first one and goes further, by detecting uniform or affine data that are provided at the input of functional units and that remains uniform or affine at the output. In that case, the result can be computed by dedicated scalar hardware like in Cray processors or by relying on the existing vector hardware. As we target power reduction, the scalar solution would provide automatic reduction. However, this solution would cause data duplication in the register file and make the operand datapath more complex. The second solution can benefit from techniques that were not available at the time Cray machines were designed, such as clock-gating. This second solution can reuse the same vector hardware, with one or two scalar units enabled to compute the result, the other units being shut down using fine-grained stage-based clock gating as in the case of the IBM Cell SPU FPU [7]. The large vector length (32) used in the Tesla architecture promises larger power reductions than observed for more conventional SIMD extensions (typical length 4).

Instructions executed by GPUs show that most of uniform and scalar data come from a broadcast of some data or a copy of the register that contains the thread identifier. For these cases, uniform and scalar detection can be done statically for once at compile time or dynamically in hardware.

A static detection involves architectural as well as microarchitectural modifications. First, each instruction and register detected as uniform or scalar by the compiler has to be tagged in the instruction word. Then at runtime during the decode stage, the hardware can automatically schedule instructions according to the tag data. A dynamic detection keeps the instruction set unchanged. However, the burden of detecting uniform and scalar data is transfered from the compiler to the hardware. This solutions requires for example a tagged vector register file.

We tested the dynamic solution based on a tagged vector register file where each tag contains the type of data stored in the associated register (uniform, affine or generic vector) using the Barra simulator. At kernel launch time, the tag of the register that contains the thread identifier is set to the affine state. Instructions that broadcast values from a location in constant or shared memory set the tag of the result to the uniform state. Tags are then propagated across arithmetic instructions according to a simple set of rules, as shown in table 1. We arbitrarily restrict the allowed strides to powers of two to allow efficient hardware implementations, and conservatively make multiplications between affine and uniform data return vectors. Additionally, the information stored in this tag may be used by memory access units as it gives information about memory access patterns.

Table 1. Examples of rules of uniform and affine tag propagation. For each operation, the first row and first column indicate the tag of the first and second operand, respectively (Uniform, Affine or Vector). The central part contains the computed tag of the result.

+	U	A	V	×	U	A	V	<<	U	A	V
U	U	A	V	U	U	V	V	U	U	A	V
A	A	V	V	A	V	V	V	A	V	V	V
V	V	V	V	V	V	V	V	V	V	V	V

Cost. A tag array contains two bits per vector register. Each multiprocessor of a NVIDIA GT200 GPU features five hundred and twelve 1024-bit registers, for a total register-file size of 512 kb (not accounting for the size of error-correction codes, if any). In a basic implementation, the associated tags would require 1 kb of extra storage, making it comparatively almost negligible.

In terms of latency, reading the tags adds one level of indirection before reading registers. In NVIDIA GPUs, registers are read in sequence for a given instruction to minimize bank conflicts [8]. Therefore, operand reads can be pipelined with tag reads. Additionally, GPUs can tolerate large instruction latencies using fast context switching between threads. The tag of the output can then be computed using a few boolean operations from the tags of the input, so the required hardware modifications are minimal. Support for broadcasting a word across all SIMD units is already available to handle operands in Constant and Shared memory.

Benefits. When an input or output operand is known to be uniform, only one lane needs to be accessed. Likewise, affine vectors v such that $v_i = x + iy$ can be encoded using the base x and the stride y. Thus, their storage requirements are only two vector lanes. This reduces the used width of the register file ports and internal buses, thus saving power.

Computing a uniform or affine result function of uniform and affine inputs can be performed using only one or two Scalar Processing (SP) units with a throughput of one cycle instead of the full SIMD width during two cycles. Indeed, most arithmetic operations on affine vectors can be reduced to operations on the base and stride.

6 Technical Issues

Some issues may limit the efficiency of the proposed method and need to be taken into account in an implementation.

Partial writes. GPUs handle branch divergence using predication. A predicated instruction does not write in every lane of its output register, keeping some of them in their previous state. In this case, even if the output value is uniform (or affine), the uniform (affine) property cannot be guaranteed for the destination register.

Half registers. The Tesla architecture allows access to lower/higher 16-bit sub-registers inside regular 32-bit registers. To handle this correctly, separate tags are needed for the lower, higher and whole parts to correctly track uniform/affine information.

Overflows. An arithmetic overflow may occur in a lane of an affine register, even if the base and stride are both representable. Overflows have no direct consequences when using two's-complement arithmetic, but casts between signed and unsigned formats of various sizes can occur, resulting for instance in an overflowing 16-bit affine value being extended into a non-affine 32-bit value.

This problem can be worked around by checking for overflows when performing affine computations, and re-issue the offending instruction as a vector operation when one is detected. Support for re-issuing instructions is already present to handle bank conflicts in the constant cache and scratchpad memory. As overflows should not occur in address calculations of correct programs, we expect it to be a rare occurrence. Indeed, we did not encounter this case in any of the benchmarks we ran.

Conversions from affine to generic vector. Instructions such as an addition involving both an affine and a generic vector may require first converting the affine input to a generic vector. As long as stride values are restricted to small powers of two, this can be implemented efficiently in hardware. However, if this situation does not appear frequently, it may be advantageous to reuse the conventional SIMD ALUs to perform the conversion, then re-issue the instruction.

7 Results and Validation

Figures 4 and 5 represent the respective proportions of uniform and affine operand captured with the proposed technique. We observe that on average, 19 % of inputs and 11 % of outputs can be identified as uniform data. These ratio go up to 34 % and 22 % respectively when considering affine data.

This means that the proposed methods reduce the bus activity between the register file and functional units for 34 % of the reads transfers. Likewise, the activity within the functional units can be reduced during 22 % of the operations executed in GPGPU computations. The power reduction brought by this technique, proportional to the activity reduction, is known to be of a critical issue for GPU [9]. Future works have to precisely quantify it.

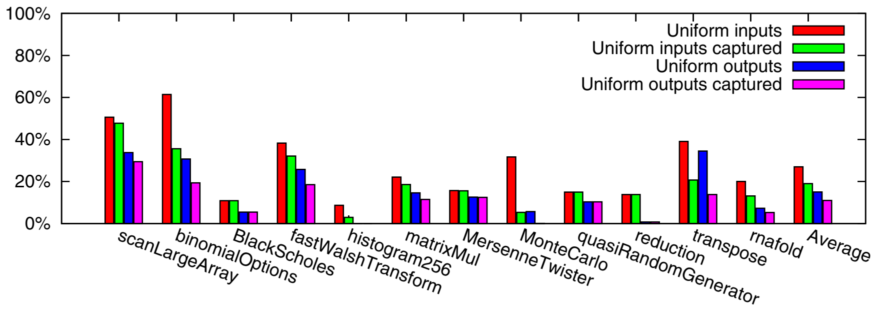


Fig. 4. Proportion of uniform operands in registers captured using our technique

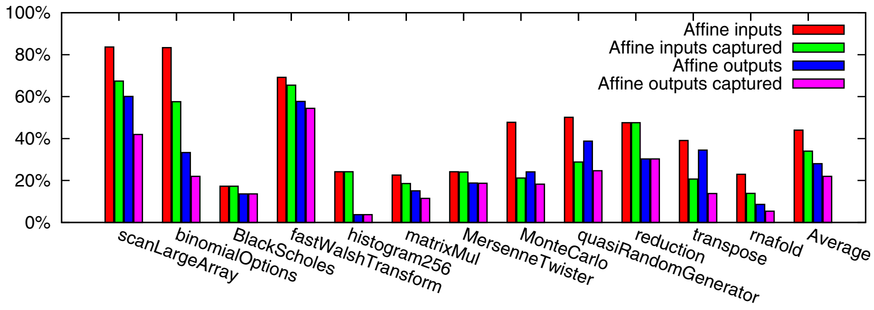


Fig. 5. Proportion of affine operands in registers captured using our technique

It can be noted that the tag technique is not optimal, as it fails to detect some uniform and affine vectors. This is mostly due to the partial write effect as described in Section 6, and complex address calculations involving multiplication, division or modulo operations. Further work may improve the accuracy of the detection.

8 Conclusion

In this paper, we presented a technique to exploit two forms of value locality specific to vector computations encountered in GPUs. The first one corresponds to the uniform pattern present when computing conditions which avoid divergence in sub-vectors. The second one corresponds to the affine pattern used to access memory efficiently. An analysis conducted on common programs used in the field of GPGPU showed that both of them are common. The novel idea of using both forms of value locality with the proposed modifications significantly reduces the power required for data transfers between the register file and the functional units as well as the power drawn by the SIMD arithmetic units. Future work will focus on improving the accuracy of the hardware-based dynamic technique presented in this article, as well as considering software-based static implementations.

Acknowledgments

We thank John Owens for his valuable comments on this work and Guillaume Rizk for the discussion related to bioinformatics applications. This work was partly supported by the French ANR BioWic.

References

1. NVIDIA: NVIDIA CUDA Compute Unified Device Architecture Programming Guide, Version 2.2 (2009)
2. Lindholm, J.E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro 28(2), 39–55 (2008)
3. Bakhoda, A., Yuan, G., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing CUDA workloads using a detailed GPU simulator. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Boston, April 2009, pp. 163–174 (2009)
4. Diamos, G., Kerr, A., Kesavan, M.: Translating GPU binaries to tiered SIMD architectures with Ocelot. Technical Report GIT-CERCS-09-01, Georgia Institute of Technology (2009)
5. Collange, C., Defour, D., Parello, D.: Barra, a parallel functional GPGPU simulator. Technical Report hal-00359342, Université de Perpignan (2009), `http://hal.archives-ouvertes.fr/hal-00359342/en/`
6. Rizk, G., Lavenier, D.: GPU accelerated RNA folding algorithm. In: Allen, G., Nabrzyski, J., Seidel, E., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2009. LNCS, vol. 5544, pp. 1004–1013. Springer, Heidelberg (2009)
7. Mueller, S., Jacobi, C., Oh, H.J., Tran, K., Cottier, S., Michael, B., Nishikawa, H., Totsuka, Y., Namatame, T., Yano, N., Machida, T., Dhong, S.: The vector floating-point unit in a synergistic processor element of a CELL processor. In: 17th IEEE Symposium on Computer Arithmetic (ARITH-17), June 2005, pp. 59–67 (2005)
8. Lindholm, E., Siu, M.Y., Moy, S.S., Liu, S., Nickolls, J.R.: Simulating multiported memories using lower port count memories. US Patent US 7339592 B2, NVIDIA Corporation (March 2008)
9. Collange, C., Defour, D., Tisserand, A.: Power Consumption of GPUs from a Software Perspective. In: Allen, G., Nabrzyski, J., Seidel, E., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2009. LNCS, vol. 5544, pp. 922–931. Springer, Heidelberg (2009)

Automatic Calibration of Performance Models on Heterogeneous Multicore Architectures

Cédric Augonnet, Samuel Thibault, and Raymond Namyst

INRIA Bordeaux, LaBRI, University of Bordeaux

Abstract. Multicore architectures featuring specialized accelerators are getting an increasing amount of attention, and this success will probably influence the design of future High Performance Computing hardware. Unfortunately, programmers are actually having a hard time trying to exploit all these heterogeneous computing units efficiently, and most existing efforts simply focus on providing tools to offload some computations on available accelerators. Recently, some runtime systems have been designed that exploit the idea of scheduling – as opposed to offloading – parallel tasks over the whole set of heterogeneous computing units. Scheduling tasks over heterogeneous platforms makes it necessary to use accurate prediction models in order to assign each task to its most adequate computing unit [2]. A deep knowledge of the application is usually required to model per-task performance models, based on the algorithmic complexity of the underlying numeric kernel.

We present an alternate, auto-tuning performance prediction approach based on performance history tables dynamically built during the application run. This approach does not require that the programmer provides some specific information. We show that, thanks to the use of a carefully chosen hash-function, our approach quickly achieves accurate performance estimations automatically. Our approach even outperforms regular *algorithmic* performance models with several linear algebra numerical kernels.

1 Introduction

Multicore architectures are now widely adopted throughout the computer ecosystem. There is also clear evidence that solutions based on specialized hardware, such as accelerator devices (*e.g.* GPGPUs) or integrated coprocessors (*e.g.* Cell's SPUs) are offering promising answers to the physical limits met by processor designers. Future processors will therefore not only get more cores, but some of them will be tailored for specific workloads.

In spite of their promising performance in terms of computational capabilities and power efficiency, such heterogeneous multicore architectures require appropriate tools. This introduces challenging problems at all levels, ranging from programming models and compilers to the design of libraries with a real support for heterogeneity. As they offer dynamic support for what has become hardly doable in a static fashion, runtime systems have a central role in this software stack. In previous work, we have therefore developed StarPU [2], a unified runtime system that offers support for heterogeneous multicore architectures. Its specificity is that it not only targets accelerators (GPUs, Cell's SPUs, *etc.*) but also multicore processors *at the same time*, in a portable fashion. StarPU also provides portable performance thanks to a high-level framework for designing portable scheduling policies.

Performance modeling is a very common technique in the scheduling literature. Whenever doable, practically building such models usually requires consequent efforts along with a certain knowledge of both the application algorithm and the underlying architecture. This is even more difficult in the case of heterogeneous platforms. But without an appropriate interface, such knowledge is not available from the runtime system's perspective: describing a task as a function pointer and pointers to the data (similar to OpenMP 3.0 tasks) does not really give much information to the runtime system in charge of the scheduling. (Un)fortunately, current accelerators reintroduce the problem of data management across a distributed memory model, so that we *have* to adopt much more expressive task APIs anyway. The majority of the programming models that target accelerators (and that do not just delegate data movements to the programmer!) require to explicitly describe which data is accessed by a task [6,7,10,1]. While this adds constraints on the programmer who has to adapt its applications to those expressive programming interfaces, the underlying runtime system gets much more information.

In this paper, we explain how StarPU takes advantage of that expressiveness to seamlessly build performance models on heterogeneous multicore architectures. Then, we illustrate how this systematic approach performs in terms of prediction accuracy and regarding its impact on the actual performance. Finally, we show that StarPU not only grabs information from the programming interface to perform better scheduling, but it also returns performance feedback information thanks to convenient tools which are helpful for instance in the context of auto-tuned libraries or when analyzing performance.

2 StarPU, a Runtime System for Heterogeneous Machines

In this section, we briefly present StarPU, our unified runtime system designed for heterogeneous multicore platforms, described in more details in a previous paper [2]. It distributes tasks onto both accelerators and processors *simultaneously* while offering portable performance thanks to generic scheduling facilities.

2.1 A Unified Runtime System

The design of StarPU is organized around three main components: a portable offloable-task abstraction, a library that manages data movements across heterogeneous platforms, and a flexible framework to design portable scheduling policies.

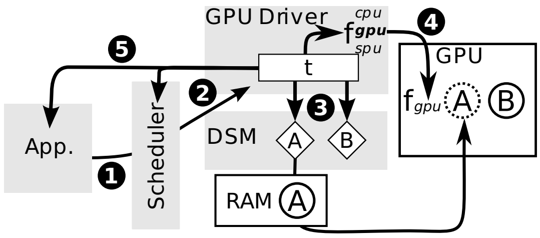


Fig. 1. Execution of a Task within StarPU. Applications submit tasks that are dispatched onto the different drivers by the scheduler. The driver offloads the computation, using the proper implementation from the codelet, and the DSM (Distribution Shared Memory) ensures the availability of coherent data. A callback is executed when the task is done.

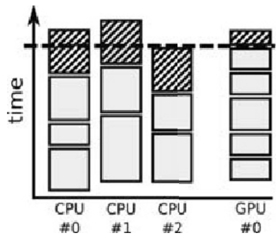


Fig. 2. The "Earliest Finish" Scheduling Strategy

A unified execution model. StarPU exposes the structure of *codelet*, which is the set of implementations of the same computation kernel (*e.g.* a vector sum) for different computation units (*e.g.* CPU and GPU). A StarPU task is then an instance of a codelet applied to some data. Figure 1 shows the path followed by tasks in StarPU. The programmer explicitly submits (graphs of) tasks to StarPU which maps them as efficiently as possible on the eligible processing units. Instead of hard-coding all the interactions between the processing units, StarPU makes it possible to concentrate on the design of efficient computational kernels and algorithmic problems instead of being stuck by low-level concerns.

A data management library. Maintaining data coherency (and availability) is a crucial issue with accelerators. In a previous paper [1], we have designed a high-level data management library that is integrated in StarPU. Mapping data statically is not necessarily sufficient when multiple processing units access the same pieces of data. The resulting data transfers are critical for the overall performance so that integrating data management within StarPU made it possible to apply optimizations (*e.g.* prefetching, reordering, asynchronous memory transfers) and to guide the scheduler.

A scheduling framework. StarPU not only executes tasks, but it also maps them as efficiently as possible thanks to its expressive scheduling interface. Hence, StarPU offers a flexible framework to implement portable scheduling policies [2]. Such policies are portable in the sense that they are directly applicable to platforms as different as a Cell processor and a hybrid GPU/CPUs machine.

2.2 Scheduling Strategies Based on Performance Models

In a previous paper [2], we have presented various scheduling policies implemented in StarPU with relatively little effort. For instance, one of these policies is similar to the HEFT scheduling strategy [12]. As shown in Figure 2, the scheduler keeps track of the expected duration until the different processing units are available. When a task is submitted to the scheduler, it is attributed to the processing unit that minimizes termination time according to the expected duration of the task on the different architectures (depicted by hatchings).

We have for instance used this rather simple strategy successfully to obtain superlinear speedups on an LU decomposition thanks to per-architecture performance models that take into account the (lack of) affinity of tasks with the different processing units. However this strategy requires that we can approximate the execution time of the tasks on the various architectures.

3 Dynamically Building Performance Models

In this section, we discuss how we can build performance models, and we give a systematic approach to dynamically construct and query a performance model based on historical knowledge, seamlessly for the programmer.

In the context of dynamic task scheduling, we do not need perfectly accurate models, but we need to take appropriate decisions when assigning the tasks onto the different processing units. Our performance models should for instance capture the relative speedups as well as the affinities between tasks and processors.

3.1 How to Define a Performance Model?

In order to define a performance model, we need to decide which parameters the model should depend on.

The most obvious parameters to describe a task are the kernel and the architecture: in the case of a matrix product on CUDA, we could for instance identify a task by the pair (SGEMM, CUDA) and associate it with its predicted execution time. A trivial refinement is to consider the total size of the tasks' data, so we can also associate this pair with a parametric cost function depending on that size (*e.g.* $\mathcal{O}\left(S^{3/2}\right)$ in the case of SGEMM applied on a matrix of size S).

The total size is often not sufficient: in the case of a kernel handling a $(n \times m)$ matrix in $\mathcal{O}(n^2 m)$, we must make a distinction between (1024×512) and (512×1024) matrices (for example). Such multivariate models are however only applicable if we have sufficient knowledge of the algorithm, which a runtime system could hardly infer automatically in a generic way. Finding an explicit model of the execution time can also be awkward because of architectural concerns such as the size of caches. Using piecewise models is possible, but it requires to delimit the boundaries of the pieces, which can be time demanding, especially for a multivariate model and in a heterogeneous environment.

In many classes of algorithms, we can reasonably make some extra regularity assumption such as most tasks handling blocks of fixed size (*e.g.* in tiled algorithms), or a limited set of sizes (*e.g.* in divide and conquer algorithms). In this case, explicitly modeling the performance as a function of the data size can be unnecessarily complicated. A history-based approach would be much simpler: instead of using a complicated multivariate model to differentiate between a (1024×512) matrix and a (512×1024) one, we simply store the execution time that was measured for those different input configurations. The advantage of this approach is that it is transparent to the programmer as long we have some mechanism to match a task with those previously executed. This method is however not applicable to irregular applications: if we know the performance of a kernel on a (1024×512) matrix, no prediction can be made for a (1026×510) matrix for instance. In the next section, we present how StarPU implements history-based models with sufficient performance feedback to help programmer easily decide whether this is an appropriate model or not.

3.2 How to Build Performance Models?

There are various ways to determine the parameters required to build the performance models that we have described in the previous section: either completely manual, or completely automated, depending on the type of the adopted model.

Building a performance model by hand (*e.g.* using the ratio between the number of operations and the speed of the processor) is hardly applicable to modern processors and require a detailed knowledge of both the application and the architecture. In the case of heterogeneous multicore processors, with multiple processing units to handle, this becomes rather unrealistic. It is however possible to design a model based on the amount of computations per task, and to calibrate the parameters by the means of a regression.

It is common to use specific precalibration programs to build those models. While this may be suited for kernels that are widely used (*e.g.* BLAS), this requires a specific test-suite and the corresponding inputs, which often represents an important programming overhead. In the context of multicore architectures, it is even harder to create a realistic workload: independently benchmarking the various processing units without taking into account the various interactions (*e.g.* cache sharing or bus contention) may not result in reliable measures.

On the other hand, it is possible to measure the performance of the different tasks during an actual execution. This does not require any additional programs, and it provides realistic performance measurements. StarPU can therefore automatically calibrate parametric models, either at runtime using linear regression models (*e.g.* of the form $\mathcal{O}(n^\alpha)$) or offline in the case of non-linear models (*e.g.* of the form $\alpha n^\beta + \gamma$, as shown in Figure 7). StarPU also builds history-based performance models by storing the performance of the tasks on different inputs, transparently for the application.

3.3 A Generic Approach for Building History-Based Performance Models Dynamically

This section shows how StarPU keeps track of the performance obtained by the tasks on the different input, and how it is possible to match a task with its similar predecessors. As shown in Figure 3, this process involves three main steps: measuring the actual duration of the tasks when they are executed and

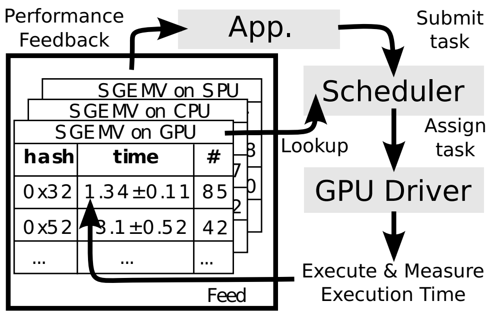


Fig. 3. Performance feedback loop

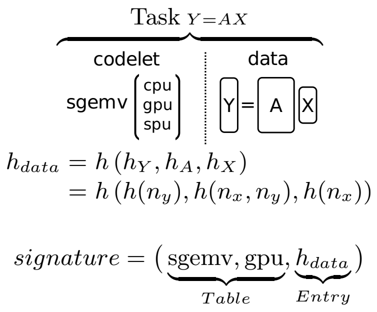

Task $Y{=}AX$
codelet
data
sgemv
cpu
gpu
spu
Y = A X
$h_{data} = h\left(h_Y, h_A, h_X\right)$
$= h\left(h(n_y), h(n_x, n_y), h(n_x)\right)$
$signature = \left(\underbrace{\text{sgemv}, \text{gpu}}_{Table}, \underbrace{h_{data}}_{Entry}\right)$

Fig. 4. Uniquely identifying a task

integrating these measurements in the history log of the task; being able to lookup the performance of some task according to the previous measurements; and offering some performance feedback to the application.

Measuring tasks' duration. Measuring the time spent to compute a task is usually simple thanks to the cycle counter facility provided by most manufacturers. In the case of Cell processors, which lacks such functionality, we had to make the SPUs transmit those measurements to the PPU along with the output data, this is not intrusive since DMA transfers are overlapped.

Identifying task kinds. We use the layout and size of the data to distinguish the different kind of instances of a computational kernel. We now explain how to compute a hash value to characterize the data layout of a task.

StarPU's data management library not only manipulates buffers described by a pointer and its length, but it also handles a mixture of various high-level data *interfaces* [1]. In Figure 4, a matrix-vector product accesses a set of matrices and vectors. There can also be much more complex data interfaces (*e.g.* compressed sparse matrices), but the size of any piece of data must be characterized by a k-tuple of parameters $(p_1, \ldots, p_k)$ where k and the parameters depend only on the data interface. A matrix is for instance described by a pair (n, m), and a single parameter is sufficient to describe the length of a vector.

We now define a hash function that computes a unique identifier for such a set of parameters. As shown in Figure 4, we characterize the size of each piece of data by applying a hash function[1] to the parameters $p_1, \ldots, p_{k-1}$ describing it. By then applying the hash function to the different per-data hashes, we get a characterization of the data layout and size for the whole task. Applying this method on a tiled algorithm would for instance result in having as many hash values as there are tile sizes.

Feeding and looking up from the model. It is now extremely simple to implement a model based on the history in StarPU: each computational kernel

[1] For example, we can use the usual CRC hash functions: $h(p_1, \ldots, p_k) = CRC(p_1, \ldots, CRC(p_{k-1}, CRC(p_k, 0)))$.

is associated with a hash table per architecture. When a task is submitted to StarPU, it computes its hash, and consults the hash table corresponding to the proper kernel-architecture pair to retrieve the average execution time previously measured for this kind of task. The average execution time and other metrics such as standard deviation are updated when a new measure is available. Hash tables can be saved (or loaded) to (from) a file so that these performance models are persistent between different runs. We therefore rapidly calibrate models by running small problems that have the same granularity as the actual problems.

4 Experimental Validation

We have implemented these automatic model calibration mechanisms in StarPU which runs on multicore CPUs, GPUs and Cell processors. In this section, we give evidence that they have a significant impact on performance; we also illustrate the performance feedback offered by StarPU, and how StarPU provides some tools to help the programmer to understand the obtained performance, and to select the most appropriate models in consequence. We here show how these mechanisms perform in the case of a hybrid platform with a nVidia Quadro FX4600 GPU and a E5410 Xeon quad-core CPU.

4.1 Sharpness of the Performance Prediction

Figure 5 shows the results obtained on an LU decomposition for two different problem sizes. The first line exhibits the average and standard deviation of the reference performance obtained when using a greedy scheduling policy to distribute tasks to CPUs and the GPU. The second line shows the results obtained when calibrating the history-based performance model after either one, two or three runs and the average performance (and standard deviation) obtained after 4 runs. During the first execution, the greedy strategy clearly outperforms the non-calibrated strategy based on performance models. But once the model is calibrated, the performance obtained by the model-based strategy gets better, not only in terms of average speed, but also with respect to the standard deviation.

		Speed (GFlop/s)	
Policy	Size	$(16k \times 16k)$	$(30k \times 30k)$
Greedy (avg.)		89.98 ± 2.97	130.68 ± 1.66
Perf. Model	1st iter.	48.31	96.63
	2nd iter.	103.62	130.23
	3rd iter.	103.11	133.50
	≥ 4 (avg.)	103.92 ± 0.46	135.90 ± 0.64

Fig. 5. Impact of performance sampling on the speed of an LU decomposition (in GFlop/s)

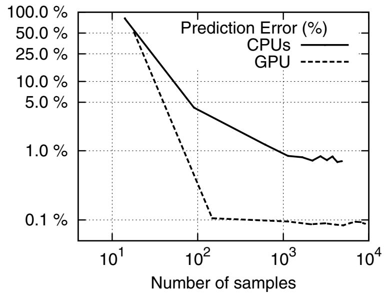


Fig. 6. Performance model accuracy

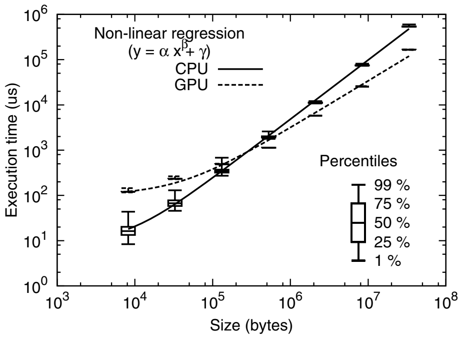


Fig. 7. Performance and regularity of an STRSM BLAS3 kernel depending on granularity

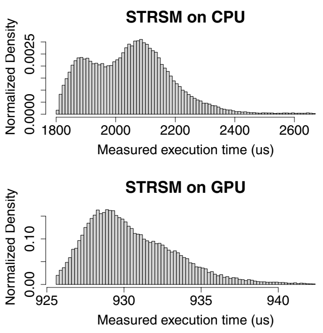


Fig. 8. Distribution of the execution times of a STRSM kernel measured for tiles of size (512 × 512)

The improvement between the runs is explained by the fact that the application runs on a hybrid CPU/GPU platform: the better the accuracy, the better the load balancing. Until the models are properly calibrated, some processing units receive too much work while others are not kept busy enough.

Figure 6 depicts the evolution of the prediction inaccuracies depending on the number of collected samples. More precisely, the error is computed by taking the sum of the absolute differences between prediction and measurements, for all tasks, and by dividing this total prediction error by the total execution time. As suggested by Figure 5, the accuracy of the models becomes better as we keep collecting measurements. We finally obtain an accuracy of an order of 1% for multicore CPUs, and below 0.1% for a GPU. This difference is due to complex interactions occuring within multicore CPUs (*e.g.* cache sharing and contention) while computations are not perturbed on GPUs. The large majority of tasks in an LU decomposition are matrix products, whose performance is especially regular even on CPUs, so that we obtain a relatively good overall accuracy.

4.2 Performance Feedback Tools

StarPU provides tools to detect tasks that are not predictable enough (*e.g.* BLAS1 kernels). Figures 7 and 8 are automatically generated by StarPU, which can collect performance measurements at runtime.

Figure 7 summarizes the behaviour of a kernel on all input sizes and the performance variations observed for the different sizes, and for the different architectures; it also shows the non-linear regression-based performance models automatically generated by StarPU so that we can figure out whether such a model is applicable or not. It also illustrates in which situation it is worth using accelerators or CPUs, therefore helping to select the most appropriate granularity. Using a small grain size on CPUs results in variable execution times, certainly explained by a poor cache use which makes performance very sensitive to the bus contention for instance. This problem disappears as we take large tiles, or if we use a GPU that is much less sensitive to such variations.

Figure 8 shows the actual distribution of the measurements that were collected for a given hash value. This not only gives a precise idea of the performance dispersion, but it can also be used to understand the actual performance issues: on the very predictable GPUs, we obtain a very thin peak, while on the CPUs, the distribution exhibiting two hills suggests that there may be some contention issue which should be further analyzed.

5 Related Works

Auto-tuning techniques have been successfully used to automatically generate the kernels of various high-performance libraries such as ATLAS [4], FFTW, OSKI or SPIRAL; and similar results are obtained in the context of GPU computing by the MAGMA project[9]. While performance models permit to generate efficient computational kernels even on heterogeneous systems, computations are usually mapped statically on the different processing resources when dealing with hybrid systems [11].

Iterative compilation frameworks also use performance feedback to take the most appropriate optimization decisions. Jimenez *et al.* [8] keep track of the relative speedups of the applications on the different architectures to decide which processing unit should be assigned to an application. Their approach is much less flexible since it does not allow to actually schedule interdependent tasks within an application.

Different runtime systems currently offer support for accelerators [3], or even hybrid systems. Similarly to StarPU, the Harmony runtime system targets hybrid platforms while proposing some scheduling facilities, possibly based on performance modeling [5]. Its performance is modeled by the means of (possibly multivariate) regression models. This approach is hardly applicable without any support from the programmer, and possibly requires a large number of samples to have a reliable model. Thanks to the high-level support for data management integrated within StarPU, the history-based solution that we propose in this paper is simpler as it is completely transparent for the programmer.

6 Conclusion

We have proposed a generic approach to seamlessly build history-based performance models. It has been implemented within the StarPU runtime system with the support of its integrated data management library, and we have shown how StarPU's performance feedback tools help the programmer to analyze whether the resulting performance prediction are relevant or not.

Such history-based performance models naturally rely on some regularity hypothesis since it cannot predict the behaviour of a task if all its predecessors had different sizes: in that case, a parametric performance model calibrated by the means of regressions is more suitable. Our history-based approach also requires computational kernel with a static flow control. Tasks' execution time should be independent from the actual content of the data, the latter is often unknown when the scheduling decisions are taken anyway. This does not require any effort from the programmer who can easily use our auto-tuning mechanisms to see whether such models results into performance improvements or not.

This technique is directly applicable to the case of complex hybrid setups (*e.g.* heterogeneous multi-GPU). This work could also be extended to model the performance of memory transfers to schedule them as well. Scheduling policies could take advantage of performance models that depend on the actual state of the underlying machine: using hardware performance counters, the history-based models could for instance keep track of contention or cache usage. Finally, performance feedback can be valuable: this not only helps to understand the behaviour of an application during a post-mortem analysis, but this is also useful for iterative compilation environments and auto-tuned libraries.

References

1. Augonnet, C., Namyst, R.: A unified runtime system for heterogeneous multicore architectures. In: Euro-Par 2008 Workshops - HPPC'08, Las Palmas de Gran Canaria, Spain (August 2008)
2. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009 Parallel Processing. LNCS, vol. 5704, pp. 863–874. Springer, Heidelberg (2009)
3. Bellens, P., Perez, J.M., Badia, R.M., Labarta, J.: Cellss: a programming model for the cell be architecture. In: Proceedings of SC'06, Tampa, Florida (2006)
4. Clint Whaley, R., Dongarra, J.: Automatically Tuned Linear Algebra Software. In: Proceedings of SIAM PP'99, San Antonio, Texas (March 1999)
5. Diamos, G., Yalamanchili, S.: Harmony: Runtime Techniques for Dynamic Concurrency Inference, Resource Constrained Hierarchical Scheduling, and Online Optimization in Heterogeneous Multiprocessor Systems. Technical report, Georgia Institute of Technology, Computer Architecture and Systems Lab (2008)
6. 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)
7. 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: Proceedings of SC'06, Tampa, Florida (2006)
8. Jiménez, V.J., Vilanova, L., Gelado, I., Gil, M., Fursin, G., Navarro, N.: Predictive runtime code scheduling for heterogeneous architectures. In: Seznec, A., Emer, J., O'Boyle, M., Martonosi, M., Ungerer, T. (eds.) HiPEAC 2009. LNCS, vol. 5409, pp. 19–33. Springer, Heidelberg (2009)
9. Li, Y., Dongarra, J., Tomov, S.: A Note on Auto-tuning GEMM for GPUs. In: Proceeding of ICCS'09, Baton Rouge, Louisiana, U.S.A. (2009)
10. McCool, M.D.: Data-Parallel Programming on the Cell BE and the GPU using the RapidMind Development Platform. In: GSPx'06 Multicore Applications Conference (2006)
11. Tomov, S., Dongarra, J., Baboulin, M.: Towards Dense Linear Algebra for Hybrid GPU Accelerated Manycore Systems. Technical report (January 2009)
12. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 13(3), 260–274 (2002)

Seventh International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar 2009)

Preface

The International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar) workshop is intended to be a forum for people working on algorithms, programming languages, tools, and theoretical models aimed at efficient problem solutions on heterogeneous platforms. The topics to be covered target heterogeneous systems and platforms and include parallel programming paradigms and models, parallel algorithms, programming languages and libraries, fault tolerance, tools for grid, cloud and green computing, and the usage of these complex platforms for solving the different type of problems and applications.

Heteropar 2009 is the seventh edition of this workshop, but it is the first year that we have the workshop co-located with the Euro-Par conference. Out of 24 manuscripts submitted this year, 10 were accepted for presentation at the workshop in Delft in August 25, corresponding to an acceptance rate of 41%. Submissions have been prepared from authors from 14 countries and received 4 reviews each from members of the Program Committee. Apart from the presentations of the 10 accepted papers, the workshop had one invited speaker of international reputation, Dick Epema, who discussed different forms of heterogeneity in large-scale systems, namely: the problem of processor co-allocation in multi-cluster systems with their heterogeneous local and wide-area networks; the exploitation of different social roles in cooperative downloading in BitTorrent-based peer-to-peer systems.

The paper entitled "static worksharing strategies for heterogeneous computers with unrecoverable failures" extends classic results from divisible load theory for master-slave platforms to optimally distribute a given workload to computers that may differ in speed and subject to interruptions of known likelihood, that may kill all work in progress on one of these computers.

The "resource allocation for multiple concurrent in-network stream-processing applications" paper analyses the operator-mapping problem for multiple concurrent in-network stream-processing applications, by attempting to use as few resources as possible for a given Quality of Service (QoS) requirement of the application. This paper has received the best paper award of the workshop.

The paper entitled "distributed data partitioning for heterogeneous processors based on partial estimation of their functional performance models" presents a new algorithm to perform optimal data partitioning on parallel computers with heterogeneous processors. Instead of assuming the speed functions to be given, the proposed algorithm uses a computational kernel to estimate the speed functions of the processors for different problem sizes during its execution.

The "two-dimensional matrix partitioning for parallel computing on heterogeneous processors based on their functional performance models" paper addresses 2D matrix partitioning for parallel computing on heterogeneous processors based on their functional performance models. The presented experimental results showed the efficiency of the proposed partitioning algorithm for matrix multiplication.

"An efficient weighted bi-objective scheduling algorithm for heterogeneous systems" paper proposes the Makespan and Reliability Cost Driven (MRCD) heuristic, a static scheduling strategy for heterogeneous distributed systems that not only minimizes the makespan but also maximizes the reliability of the application. Instead of considering both objectives in a hierarchical fashion, the proposed cost function integrates both objectives.

The "accelerating S3D: a GPGPU case study" paper focuses on experiences from accelerating S3D, a high-fidelity turbulent reacting flow solver, on graphics processors. This paper also addresses the issue of floating point accuracy and precision on the GPU, a topic of enormous importance to scientific computing.

The paper "using hybrid CPU-GPU platforms to accelerate the computation of the matrix sign function" investigates the numerical computation of the matrix sign function of large-scale dense matrices in heterogeneous architectures, such as a current general-purpose multi-core processor connected to a graphics processor. Parallelism is extracted in both processors by linking sequential versions of the codes with multi-threaded implementations of BLAS.

The "modelling pilot-job applications on production grids" paper presents a performance model for pilot-job applications running on production grids. Statistics are derived about the number of available pilots along time and the makespan of the application given the number of submitted pilots. Results obtained on a radiotherapy application running on the EGEE production grid show the accuracy of the model to correctly describe the behavior of the application.

The paper "modeling resubmission in unreliable grids: the bottom-up approach" analytically and experimentally studies resubmission in the case of random brokering, by considering that jobs can be resubmitted to the broker or to the computing. The obtained results show that resubmits to the broker is a better strategy.

The "reliable parallel programming model for distributed computing environments" paper presents a trustworthy programming model for grid computing environments that achieves reliability in a transparent manner for the programmer. It is based on an active replication scheme, capable of supporting arbitrary fail-silent and fail-stop node failures.

As Program Chair I wish to acknowledge all those that contributed to the success of HeteroPar 2009, in particular to the authors of the submitted papers, and to the Program Committee members for their valuable time and expertise to the selection process.

October 2009

Leonel Sousa
Program Chair
HeteroPar 2009

Static Worksharing Strategies for Heterogeneous Computers with Unrecoverable Failures

Anne Benoit[1,4], Yves Robert[1,4], Arnold Rosenberg[2], and Frédéric Vivien[3,4]

[1] Ecole Normale Supérieure de Lyon, France
{Anne.Benoit,Yves.Robert,Frederic.Vivien}@ens-lyon.fr
[2] Colorado State University, Fort Collins, USA
rsnbrg@colostate.edu
[3] INRIA, France
[4] LIP, UMR 5668 ENS-CNRS-INRIA-UCBL, Lyon, France

Abstract. One has a large workload that is "divisible" (its constituent work's granularity can be adjusted arbitrarily) and one has access to p remote computers that can assist in computing the workload. How can one best utilize the computers toward this end? Two features complicate this question. First, the remote computers may differ from one another in speed. Second, each remote computer is subject to interruptions of known likelihood that kill all work in progress on it. One wishes to orchestrate sharing the workload with the remote computers in a way that maximizes the expected amount of work completed, given the risk of interruptions. We consider three versions of the preceding problem. Two versions envision *heterogeneous* computing resources: the remote computers may differ from one another in speed; one version envisions *homogeneous* computing resources: the remote computers are identical. One of the heterogeneous versions ignores communication costs (i.e., assumes that they are negligible); the other two versions account explicitly for communication costs. We provide exact expressions for the optimal work expectation for all three versions of the problem. For the most general version (heterogeneous resources, with communication costs), we provide a recurrence for computing this expectation; for the other two versions, we provide closed-form expressions.

1 Introduction

This paper extends well-known results from divisible load theory [11] concerning master-worker computing platforms. Our goal is to optimally distribute a large workload to p remote computers (the "workers") that may differ in speeds; the workers are connected to the "master" via a bus or network. The master wants to send a fraction of the load to each worker, seriatim. The problem is to determine, for each worker, the fraction of the load that it should be sent and the order in which it should be served. This problem has received considerable attention in recent years, and closed-form expressions have been derived for these load fractions [7, 4]. We revisit this problem in the context of remote computers that are subject to *unrecoverable interruptions* [5], and we aim to maximize the expected amount of total work that will be completed. For intuition: An "unrecoverable interruption" may correspond to a hardware crash, an event that is increasingly likely with the advent of massively parallel grid platforms [1, 2]; it may also correspond to the unexpected return of a remote computer's user/owner during an episode of cycle-stealing [3, 10, 12]. Consider the following scenario: on Friday evening, a PhD student has a large set of simulations to run. S/he has access to a set of computers from the lab, but each computer can be reclaimed at any instant by its owner. In any case, everybody will be back to work on Monday 8am. What is the student's best strategy? How much simulation data should s/he send to, and execute on, each accessible computer?

We cleave to the preceding scenario and assume that remote computers are vulnerable to interruption, with a risk that grows *linearly with the time the computer has been available.* Other probability distributions can be envisioned, but the linear distribution is very natural in the absence of further information. Also, the linear risk function turns out to be tractable: we have derived optimality results for this distribution. The major achievement of this paper is to provide a distribution strategy that maximizes the expected total amount of work completed.

The paper is organized as follows. We describe the formal framework in detail, in Section 2. We then address three optimization problems. The simplest problem ignores communication costs, but considers a heterogeneous set of resources that may differ in speed (Section 3). The other two problems account for communication costs, first with identical remote computers (Section 4) and then with computers that may differ in speed (Section 5). We provide exact expressions for the optimal work expectation for all three problems. For the first two problems we provide explicit, closed-form expressions; for the last (and most general) problem, we have to resort to a complicated recurrence formula that yields the optimal solution for specific instances in linear time. We provide a brief overview of related work in Section 6; in particular, we compare our approach and results with those of our previous work [5]. We close with some conclusions and perspectives in Section 7.

2 Framework

We have W units of divisible work to execute on p remote computers. Each computer is susceptible to unrecoverable interruptions that "kill" all work in progress (on that computer). All remote computers share the same perfectly known instantaneous probability of being interrupted, and this probability increases with the amount of time the computer has been operating (whether working or not). Within our model, all computers obey the same *risk function* $Pr(T)$ of having been interrupted by the end of the first T time units.

The risk function that is the focus of our study is the linear function $Pr(w) = \kappa w$. It is the most natural model in the absence of further information: the risk of interruption grows linearly with the time that the computer has been available, or equivalently with the amount of work that it could have done. The density function is then $dPr = \kappa dt$ for $t \in [0, 1/\kappa]$ and 0 otherwise, so that

$$Pr(T) = \min\left\{1, \int_0^T \kappa dt\right\} = \min\{1, \kappa T\}.$$

We assume that all p computing remote computers obey the same probability failure distribution. For instance, in the earlier-mentioned cycle-stealing scenario, the remote computers are computers from the CS department that can be loaned during the week-end, so they have the same probability of having their owner returning. With more information about the owners, we could refine the scenario and assume different laws for, say, students and staff.

The speed of computer P_i is speed_i. The computers are interconnected by a bus or homogeneous network of bandwidth bw. Each computer will receive a single message from the master that contains its *work chunk*, i.e., all the data necessary to execute its assigned work: in the terminology of [7], this is a single-round distribution strategy. Work is transmitted sequentially to each remote computer, as in the standard divisible load model of [7]. This corresponds to a (somewhat pessimistic) *one-port* model [9], with single-threaded execution and blocking send/receive MPI primitives [13]. We introduce two important notations:

– $\mathsf{z} = \frac{\kappa}{\mathsf{bw}}$, the failure-rate per unit-load communication from the master to any computer;
– $\mathsf{x}_i = \frac{\kappa}{\mathsf{speed}_i}$, the failure-rate per unit-load computation by computer P_i.

If we send a load w_1 to computer P_1 and then a load w_2 to computer P_2, then the expected amount of work executed by P_1 is

$$E_1 = w_1 \left(1 - (\mathsf{z} + \mathsf{x}_1) w_1\right). \tag{1}$$

To understand (Eq. 1), simply observe that P_1 is communicating during the first w_1/bw time-units and is computing during the next w_1/speed_1 time-units. P_1's risk of being interrupted increases linearly with elapsed time, regardless of whether it is communicating or computing. Similarly, we derive that the expected amount of work executed by P_2 is

$$E_2 = w_2 \left(1 - \mathsf{z}(w_1 + w_2) - \mathsf{x}_2 w_2\right).$$

Indeed, P_2 starts computing only after both communications from the master to P_1 and P_2 have completed, which takes $(w_1 + w_2)/\mathsf{bw}$ time-steps; then it computes during w_2/speed_2 additional time-steps. Again, P_2's risk of being interrupted increases linearly with elapsed time, regardless of whether it is waiting (while the master communicates to P_1), or communicating, or computing. If we had only these two remote computers ($p = 2$), then our goal would be to maximize $E_1 + E_2$, the expected total amount of work done.

Note that the formula for expectation E_1 (Eq. 1) assumes that $(\mathsf{z}+\mathsf{x}_1)w_1 \leq 1$. If this condition is not satisfied, then $E_1 = 0$. To avoid such cases, we make a technical assumption and assume that the total load is small enough so that we distribute it entirely to the p computers. Indeed, if the total load is too large, then all computers will be interrupted with certainty (probability 1) before they complete their chunk. In the following, we assume that the p chunks received by the computers *partition* the original load, and that there is a nonzero probability that the last computer is not interrupted before or during its computation. A sufficient condition for this latter condition to hold is $W \leq \frac{1}{\mathsf{z}+\mathsf{x}_{\max}}$, where $\mathsf{x}_{\max} = \frac{\kappa}{\min_{1\leq i\leq p} \mathsf{speed}_i}$ is the failure-rate per unit-load computation of the slowest computer. To see this, simply note that the last computer, say P_i, can always start computing at time-step Y/bw, where $Y \leq W$ is the total load sent to all preceding computers: introducing idle times in the communication cannot improve the solution, as the risk of interruption grows with time. Then P_i needs V/speed_i time-steps to execute its own chunk of size V, where $Y + V \leq W$, whence the claim. We can now formally state the optimization problem:

Definition 1. *We let* DISTRIB(p) *denote the problem of computing* $\mathcal{E}^{opt}(W,p)$, *the optimal value of the expected total amount of work completed when partitioning and distributing the entire workload* $W \leq \frac{1}{\mathsf{z}+\mathsf{x}_{\max}}$ *to the* p *remote computers.*

We turn now to the case $\mathsf{z} = 0$, wherein communication costs can be neglected. Then we shall tackle the case with communication costs and identical remote computers ($\mathsf{x}_i = \mathsf{x}$ for $1 \leq i \leq p$) before moving on to the general case with communication costs and different-speed remote computers.

3 Heterogeneous Workers, No Communication Costs

In this section we deal with the DISTRIB problem when we have p remote computers that differ in speed, but we ignore communication cost (the case $\mathsf{z} = 0$). This scenario models situations wherein computations dominate in the application. We need to introduce symmetric functions to state our result.

Definition 2. *For integers* $n \geq 1$ *and* $i \in \{0, \ldots, n\}$, *we denote by* $\sigma_i^{(n)}$ *the* i*-th symmetric function of* $\mathsf{x}_1, \ldots, \mathsf{x}_n$: $\sigma_i^{(n)} = \sum_{1\leq j_1<\cdots<j_i\leq n} \prod_{k=1}^{i} \mathsf{x}_{j_k}$. *By convention* $\sigma_0^{(n)} = 1$.

For instance with $n = 3$, $\sigma_1^{(3)} = \mathsf{x}_1 + \mathsf{x}_2 + \mathsf{x}_3$, $\sigma_2^{(3)} = \mathsf{x}_1\mathsf{x}_2 + \mathsf{x}_1\mathsf{x}_3 + \mathsf{x}_2\mathsf{x}_3$ and $\sigma_3^{(3)} = \mathsf{x}_1\mathsf{x}_2\mathsf{x}_3$.

Theorem 1. *When* $\mathsf{z} = 0$ *the optimal solution to* DISTRIB(p) *sends a chunk of size* $\frac{\prod_{k\neq i} \mathsf{x}_k}{\sigma_{p-1}^{(p)}} W = \frac{\sigma^{(p)}}{\mathsf{x}_i \sigma_{p-1}^{(p)}}$ *to computer* P_i. *In this case,*

$$\mathcal{E}^{opt}(W,p) = W - \frac{\sigma_p^{(p)}}{\sigma_{p-1}^{(p)}} W^2 = W - \frac{1}{\sum_{i=1}^{p} \frac{1}{\mathsf{x}_i}} W^2.$$

Proof. Let $\alpha_{i,p} = \frac{\prod_{k\neq i} \mathsf{x}_k}{\sigma_{p-1}^{(p)}}$ and $\mathsf{f}_p = \frac{\sigma_p^{(p)}}{\sigma_{p-1}^{(p)}}$. We show the result by induction. Note that it holds for $p = 1$, because $\alpha_{1,1} = 1$ and $\mathsf{f}_1 = \mathsf{x}_1$.

To help the reader follow the derivation, we prove the result for $p = 2$ before dealing with the general case. Assume that the size of the chunk sent to P_1 is Y. The size of the chunk sent to P_2 is thus $W - Y$. Both chunks are sent in parallel, as no cost is assessed for communications. The expected amount of work done is

$$E(Y) = Y\left(1 - \mathsf{x}_1 Y\right) + (W - Y)\left(1 - \mathsf{x}_2(W - Y)\right).$$

We rewrite $E(Y) = W - \mathsf{x}_2 W^2 - (\mathsf{x}_1 + \mathsf{x}_2)Y^2 + 2\mathsf{x}_2 WY$. The optimal value is $Y^{(\text{opt})} = \frac{\mathsf{x}_2}{\mathsf{x}_1+\mathsf{x}_2} W = \alpha_{1,2} W$ as desired (and $W - Y^{(\text{opt})} = \frac{\mathsf{x}_1}{\mathsf{x}_1+\mathsf{x}_2} W = \alpha_{2,2} W$). Importing the value of $Y^{(\text{opt})}$ into the expression of $E(Y)$, we derive that

$$\mathcal{E}^{\text{opt}}(W, 2) = E(Y^{(\text{opt})}) = W - \mathsf{f}_2 W^2,$$

where

$$\mathsf{f}_2 = \mathsf{x}_2 - \frac{\mathsf{x}_2^2}{\mathsf{x}_1 + \mathsf{x}_2} = \frac{\mathsf{x}_1 \mathsf{x}_2}{\mathsf{x}_1 + \mathsf{x}_2} = \frac{\sigma_2^{(2)}}{\sigma_1^{(2)}}.$$

This proves the claim for $p = 2$.

Assume now that the result holds for $\leq n$ workers. Consider the case of $n + 1$ workers, and assume that the size of the chunk sent to P_{n+1} is $W - Y$. By induction, the optimal expected amount of work done by the first n computers is $\mathcal{E}^{\text{opt}}(Y, n) = Y\left(1 - \mathsf{f}_n Y\right)$, and this is achieved by sending a chunk of size $\alpha_{i,n} Y$ to P_i for $1 \leq i \leq n$. The expected amount of work done by the $n + 1$ workers is then

$$E(Y) = Y\left(1 - \mathsf{f}_n Y\right) + (W - Y)\left(1 - \mathsf{x}_{n+1}(W - Y)\right).$$

We proceed as in the preceding case. The optimal value is $Y^{(\text{opt})} = \frac{\mathsf{x}_{n+1}}{\mathsf{f}_n + \mathsf{x}_{n+1}} W$, and we derive that $\mathcal{E}^{\text{opt}}(W, n + 1) = E(Y^{(\text{opt})}) = W - \mathsf{f}_{n+1} W^2$ where

$$\mathsf{f}_{n+1} = \mathsf{x}_{n+1} - \frac{\mathsf{x}_{n+1}^2}{\mathsf{f}_n + \mathsf{x}_{n+1}}.$$

We recognize that $\sigma_n^{(n)} + \mathsf{x}_{n+1}\sigma_{n-1}^{(n)} = \sigma_n^{(n+1)}$ so that $\mathsf{f}_n + \mathsf{x}_{n+1} = \frac{\sigma_n^{(n+1)}}{\sigma_{n-1}^{(n)}}$, and

$$\mathsf{f}_{n+1} = \mathsf{x}_{n+1} - \frac{\mathsf{x}_{n+1}^2 \sigma_{n-1}^{(n)}}{\sigma_n^{(n+1)}} = \frac{\mathsf{x}_{n+1}\left(\sigma_n^{(n+1)} - \mathsf{x}_{n+1}\sigma_{n-1}^{(n)}\right)}{\sigma_n^{(n+1)}} = \frac{\mathsf{x}_{n+1}\sigma_n^{(n)}}{\sigma_n^{(n+1)}} = \frac{\sigma_{n+1}^{(n+1)}}{\sigma_n^{(n+1)}}$$

as desired. Also, $Y^{(\text{opt})} = \frac{\mathsf{x}_{n+1}}{\mathsf{f}_n + \mathsf{x}_{n+1}} W = \frac{\mathsf{x}_{n+1}\sigma_{n-1}^{(n)}}{\sigma_n^{(n+1)}} W$. By induction, for $1 \leq i \leq n$, we get $\alpha_{i,n+1} = \alpha_{i,n} \frac{\mathsf{x}_{n+1}\sigma_{n-1}^{(n)}}{\sigma_n^{(n+1)}} = \frac{\mathsf{x}_{n+1}\sigma_{n-1}^{(n)} \prod_{1 \leq k \leq n, k\neq i} \mathsf{x}_k}{\sigma_{n-1}^{(n)}\sigma_n^{(n+1)}} = \frac{\mathsf{x}_{n+1} \prod_{1 \leq k \leq n, k\neq i} \mathsf{x}_k}{\sigma_n^{(n+1)}} = \frac{\prod_{1 \leq k \leq n+1, k\neq i} \mathsf{x}_k}{\sigma_n^{(n+1)}}$ as desired. It remains to check the value of $\alpha_{n+1,n+1} = 1 - \frac{\mathsf{x}_{n+1}\sigma_{n-1}^{(n)}}{\sigma_n^{(n+1)}} = \frac{\sigma_n^{(n+1)} - \mathsf{x}_{n+1}\sigma_{n-1}^{(n)}}{\sigma_n^{(n+1)}} = \frac{\prod_{1 \leq k \leq n} \mathsf{x}_k}{\sigma_n^{(n+1)}}$ which concludes the proof. □

We see that the optimal solution is *symmetric*: each worker's "contribution" to the expectation is a (somewhat complicated, but) symmetric function of all computer speeds.

4 Homogeneous Workers, with Communication Costs

We move now to the case with communication costs. Before dealing with the general case of heterogeneous remote computers, which turns out to be difficult, we address the homogeneous problem, with identical remote computers:

Theorem 2. *When* $\mathsf{x}_i \equiv \mathsf{x}$ *(i.e., speeds are identical), the optimal solution to* Distrib(p) *distributes equal-size chunks to the workers, i.e., chunks of size* W/p. *In this case,*

$$\mathcal{E}^{opt}(W,p) = W - \frac{(p+1)\mathsf{z} + 2\mathsf{x}}{2p} W^2.$$

Proof. The proof is similar to that of Theorem 1. Let $\mathsf{f}_p = \frac{(p+1)\mathsf{z}+2\mathsf{x}}{2p}$. We show the result by induction. Note that it holds for $p = 1$, because $\mathsf{f}_1 = \mathsf{z} + \mathsf{x}$.

Assume that the result holds for $\leq n$ workers. Consider the case of $n + 1$ workers, and assume that the size of the chunk sent to P_{n+1} is $W - Y$. By induction, the optimal expected amount of work done by the first n workers is $\mathcal{E}^{\rm opt}(Y, n) = Y(1 - \mathsf{f}_n Y)$, and this is achieved by sending a chunk of size Y/n to each P_i, for $1 \leq i \leq n$. The expected amount of work done is then

$$E(Y) = Y(1 - \mathsf{f}_n Y) + (W - Y)(1 - \mathsf{z}W - \mathsf{x}(W - Y)).$$

To understand the value of the contribution of P_{n+1} to $E(Y)$, simply note that it has to wait for the whole workload to be distributed (accounted for by the term $\mathsf{z}W$) before it can start computing its own chunk (accounted for by the term $\mathsf{x}(W - Y)$). We rewrite $E(Y)$ as

$$E(Y) = W - (\mathsf{z} + \mathsf{x})W^2 - (\mathsf{f}_n + \mathsf{x})Y^2 + (\mathsf{z} + 2\mathsf{x})WY.$$

The optimal value is $Y^{(\rm opt)} = \frac{\mathsf{z}+2\mathsf{x}}{2(\mathsf{f}_n+\mathsf{x})}W$ and we derive that $\mathcal{E}^{\rm opt}(W, n + 1) = E(Y^{(\rm opt)}) = W - \mathsf{f}_{n+1}W^2$, where $\mathsf{f}_{n+1} = \mathsf{z} + \mathsf{x} - \frac{(\mathsf{z}+2\mathsf{x})^2}{4(\mathsf{f}_n+\mathsf{x})}$.

Using the induction hypothesis, we get $\mathsf{f}_n + \mathsf{x} = \frac{(n+1)\mathsf{z}+2\mathsf{x}}{2n} + \mathsf{x} = \frac{(n+1)(\mathsf{z}+2\mathsf{x})}{2n}$, so that

$$\mathsf{f}_{n+1} = \mathsf{z} + \mathsf{x} - \frac{n(\mathsf{z} + 2\mathsf{x})}{2(n + 1)} = \frac{(n + 2)\mathsf{z} + 2\mathsf{x}}{2(n + 1)}$$

as expected. We also obtain $Y^{(\rm opt)} = \frac{n}{n+1}W$, so that each P_i (with $i \leq n$) receives a chunk of size $\frac{Y^{(\rm opt)}}{n} = \frac{W}{n+1}$. We deduce that P_{n+1} receives a chunk of that same size (or we can directly check that $W - Y^{(\rm opt)} = \frac{W}{n+1}$). □

Interestingly, the optimal solution mandates sending *equal-size* chunks to all workers. This contrasts with the standard divisible load setting. In that setting, when one aims to minimize the total time needed to execute a certain amount of work, one has all workers terminate their computations simultaneously [7], so that the earlier workers served by the master receive larger chunks than do the later workers.

5 Heterogeneous Workers, with Communication Costs

We are now ready for the general case, which accounts for communication costs while serving different-speed computers. We need a few notations before stating the main result of this paper:

Definition 3. *Define the sequence* $\lambda = \lambda_0, \lambda_1, \ldots,$ *as follows:*
$\lambda_0 = \lambda_1 = 4$; *for* $n \geq 2$, $\lambda_n = \lambda_{n-1} - \frac{1}{4}\lambda_{n-2}$
For convenience, let $\lambda_{-1} = 0$.

Note that $\lambda_n = 4(1+n)/2^n$ for all $n \geq 0$. We use the sequence λ is used to characterize the optimal solution to the general problem.

Theorem 3. *In the general case, the optimal solution to* DISTRIB(p) *does not depend upon the ordering of the communications from the master. When using the ordering* $P_1, P_2, \ldots, P_p$, *the optimal solution sends a chunk of size* $\alpha_{i,p}W$ *to* P_i, *where*

1. $\alpha_{p,p} = \dfrac{2\mathsf{f}_{p-1} - \mathsf{z}}{2(\mathsf{f}_{p-1} + \mathsf{x}_p)}$ *for* $p \geq 2$ *and* $\alpha_{1,1} = 1$;
2. $\alpha_{i,p} = \alpha_{i,i}\dfrac{\mathsf{z} + 2\mathsf{x}_{i+1}}{2(\mathsf{f}_i + \mathsf{x}_{i+1})}$ *for* $p-1 \geq i \geq 2$;
3. $\alpha_{1,p} = 1 - \alpha_{2,p}$ *for* $p \geq 2$.

In this case, $\mathcal{E}^{opt}(W,p) = W - \mathsf{f}_p W^2$, *where* $\mathsf{f}_p = \dfrac{\sum_{i=0}^{p} \lambda_i \sigma_{p-i}^{(p)} \mathsf{z}^i}{\sum_{i=0}^{p-1} \lambda_i \sigma_{p-i-1}^{(p)} \mathsf{z}^i}$ *for* $p \geq 1$

Proof. The proof is similar to those of Theorems 1 and 2 but it is more involved. Due to lack of space, we refer to the companion research report [6] for a complete proof. Note that the theorem holds for $p = 1$, because $\mathsf{f}_1 = \frac{\lambda_0 \mathsf{x}_1 + \lambda_1 \mathsf{z}}{\lambda_0} = \mathsf{z} + \mathsf{x}_1$.

To give intuition for the result, in particular why the ordering of the communications is not important, consider the case with two workers, P_1 and P_2, that are served in this order (first P_1, then P_2). If we send a chunk of size Y to P_1 and one of size $W - Y$ to P_2, we derive that the expected amount of work completed is

$$E(W) = Y(1 - \mathsf{f}_1 Y) + (W - Y)(1 - (\mathsf{z}W + \mathsf{x}_2(W - Y))).$$

As before, to understand this reckoning, we note that P_2 waits for the first chunk to be sent to P_1; then it receives its own chunk; both steps account for the term $\mathsf{z}W$ on the righthand side. Finally, P_2 computes its chunk, whence the term $\mathsf{x}_2(W - Y)$. We rewrite

$$E(Y) = W - (\mathsf{z} + \mathsf{x}_2)W^2 - (\mathsf{f}_1 + \mathsf{x}_2)Y^2 + (\mathsf{z} + 2\mathsf{x}_2)WY.$$

The optimal value is $Y^{(\mathrm{opt})} = \frac{\mathsf{z}+2\mathsf{x}_2}{2(\mathsf{f}_1+\mathsf{x}_2)}W = \alpha_{1,2}W$, and we derive that

$$\mathcal{E}^{\mathrm{opt}}(W,2) = W - \mathsf{f}_2 W^2$$

where $\mathsf{f}_2 = \mathsf{z} + \mathsf{x}_2 - \frac{(\mathsf{z}+2\mathsf{x}_2)^2}{4(\mathsf{f}_1+\mathsf{x}_2)}$. After some easy manipulation, we see that

$$\mathsf{f}_2 = \frac{4\mathsf{x}_1\mathsf{x}_2 + 4(\mathsf{x}_1 + \mathsf{x}_2)\mathsf{z} + 3\mathsf{z}^2}{4(\mathsf{x}_1 + \mathsf{x}_2 + \mathsf{z})},$$

as desired. We see that the formula is symmetric in x_1 and x_2, thereby showing that the ordering of the communications has no significance. Please refer to [6] for the general case of the induction. □

The interested reader can check that we end up with the values of f_p and $\alpha_{i,p}$ given in Theorem 1 when $\mathsf{z} = 0$, and those given in Theorem 2 when $\mathsf{x}_i = \mathsf{x}$.

6 Related Work

The divisible-load model is a reasonable abstraction of an application made up of a large number of identical, fine-grained parallel computations. Such applications are found in many scientific areas, and we refer the reader to the survey paper [11] and the journal special issue [8] for detailed examples. Also, the divisible-load approach has been applied successfully to a variety of computing platforms, such as bus-shaped, star-shaped, and even tree-shaped platforms. Despite the extensive literature on the divisible-load model, to the best of our knowledge, the current study is the first to consider the divisible-load problem on master-worker platforms whose computers are subject to unrecoverable failures/interruptions.

Our earlier work [5], and its predecessors [3, 10, 12], also consider computers with unrecoverable failures/interruptions, but with major differences in the models. The current paper allows *heterogeneous* computers and communication costs, while [5] focuses only on identical computers without communication costs. To "compensate" for the additional complexity in the model we study here, we have restricted ourselves in this paper to scenarios where the entire workload is distributed to the remote computers, a strategy that is often suboptimal, even when scheduling a single remote computer [5]. Furthermore, we have not considered here the possible benefits of replicating the execution of some work units on several remote computers, a key tool for enhancing expected work production in [5]. Obviously, it would be highly desirable to combine the sophisticated platforms of the current study with the sophisticated algorithmics of [5]. We hope to do so in future work, in order to deal with the most general master-worker problem instances—instances that allow heterogeneous computing resources and communication costs, that do not insist that all work be distributed, and that give the scheduler the option of replicating work on multiple remote computers.

7 Conclusion

In this paper we have revisited the well-known master-worker paradigm for divisible-load applications, adding the hypothesis that the computers are subject to unrecoverable failures/interruptions. In this novel context, the natural objective of a schedule is to maximize the expected amount of work that gets completed. We have succeeded in providing either closed-form formulas or linear recurrences to characterize optimal solutions, thereby providing a nice counterpart to existing results in the classical context of makespan minimization. In particular, our demonstration that the ordering of communications has no impact on the optimal solution is a very interesting (and somewhat unexpected) result, as it shows that the scheduling problem has polynomial complexity: there is no need to explore the combinatorial space of all possible orderings.

As discussed in Section 6, we have adopted certain simplifications to the general problem we ultimately aspire to. We have insisted on distributing the entire workload to the remote computers, without replication of work. Our not allowing work replication is particularly unfortunate when contemplating environments that have access to abundant computing resources. This, then, is the first projected avenue for extending the current work. Several other extensions of this work would be desirable also, for instance: (*i*) including a start-up overhead-cost each time a computer executes a piece of work (e.g., to account for the cost of initiating a communication or a checkpointing); (*ii*) studying computers that obey not only linear, but also different risk functions (e.g., when several user categories have different probabilities of returning to reclaim their computers); (*iii*) studying risk functions that are no longer linear (e.g., standard exponential or, importantly, heavy-tailed distributions); and (*iv*) analyzing multi-round strategies, wherein each remote computer receives its share of work in several rounds. Altogether, there are many challenging algorithmic problems to address!

Acknowledgments. A. Benoit, Y. Robert, and F. Vivien are with Université de Lyon, France. Y. Robert is with the Institut Universitaire de France. F. Vivien is with INRIA, France. The work of A. Benoit and Y. Robert was supported in part by the ANR StochaGrid project. The work of A. Rosenberg was supported in part by US NSF Grants CNS-0842578 and -0905399.

References

[1] Abawajy, J.: Fault-tolerant scheduling policy for grid computing systems. In: International Parallel and Distributed Processing Symposium IPDPS 2004. IEEE Computer Society Press, Los Alamitos (2004)
[2] Albers, S., Schmidt, G.: Scheduling with unexpected machine breakdowns. Discrete Applied Mathematics 110(2-3), 85–99 (2001)
[3] Awerbuch, B., Azar, Y., Fiat, A., Leighton, F.T.: Making commitments in the face of uncertainty: How to pick a winner almost every time. In: 28th ACM SToC, pp. 519–530 (1996)
[4] Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Trans. Parallel Distributed Systems 16(3), 207–218 (2005)
[5] Benoit, A., Robert, Y., Rosenberg, A., Vivien, F.: Static strategies for worksharing with unrecoverable interruptions. In: IPDPS 2009, the 23rd IEEE International Parallel and Distributed Processing Symposium. IEEE Computer Society Press, Los Alamitos (2009)
[6] Benoit, A., Robert, Y., Rosenberg, A., Vivien, F.: Static worksharing strategies for heterogeneous computers with unrecoverable failures. Research Report 2009-23, LIP, ENS Lyon, France (July 2009), `graal.ens-lyon.fr/~yrobert/`

[7] Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling Divisible Loads in Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamitos (1996)

[8] Bharadwaj, V., Ghose, D., Robertazzi, T.: Divisible load theory: a new paradigm for load scheduling in distributed systems. Cluster Computing 6(1), 7–17 (2003)

[9] Bhat, P., Raghavendra, C., Prasanna, V.: Efficient collective communication in distributed heterogeneous systems. Journal of Parallel and Distributed Computing 63, 251–263 (2003)

[10] Bhatt, S., Chung, F., Leighton, F., Rosenberg, A.: On optimal strategies for cycle-stealing in networks of workstations. IEEE Trans. Computers 46(5), 545–557 (1997)

[11] Robertazzi, T.: Ten reasons to use divisible load theory. IEEE Computer 36(5), 63–68 (2003)

[12] Rosenberg, A.L.: Optimal schedules for cycle-stealing in a network of workstations with a bag-of-tasks workload. IEEE Trans. Parallel Distrib. Syst. 13(2), 179–191 (2002)

[13] Snir, M., Otto, S.W., Huss-Lederman, S., Walker, D.W., Dongarra, J.: MPI the complete reference. The MIT Press, Cambridge (1996)

Resource Allocation for Multiple Concurrent In-network Stream-Processing Applications

Anne Benoit[1], Henri Casanova[2], Veronika Rehn-Sonigo[1], and Yves Robert[1]

[1] LIP, ENS Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France
{Anne.Benoit,Veronika.Sonigo,Yves.Robert}@ens-lyon.fr
[2] University of Hawai'i at Manoa, 1680 East-West Road, Honolulu, HI 96822, USA
henric@hawaii.edu

Abstract. This paper investigates the operator mapping problem for in-network stream-processing applications. In-network stream-processing is the application of one or several trees of operators, in steady-state, to data that are continuously updated at different locations in the network. The goal is to generate final results at a desired rate. Different operator trees may share common subtrees, so that intermediate results could be reused in different applications. This work provides complexity results for different instances of the basic problem and proposes several polynomial-time heuristics. Quantitative comparison of the heuristics in simulation demonstrates the importance of mapping operators to appropriate processors, and allows us to identify a heuristic that achieves good results in practice.

1 Introduction

We consider applications structured as trees of operators, where leaves correspond to basic data objects distributed in a network. Each internal node in the tree denotes the aggregation and combination of the data from its children, which in turn generates new data that is used by the node's parent. The computation is complete when all operators have been applied up to the root node, thereby producing a final result. We consider the scenario in which the basic data objects are constantly being updated, meaning that the tree of operators must be applied continuously. The goal is to produce final results at some desired rate.

The above problem is called *stream processing* [1] and arises in several domains. One such domain is the acquisition and refinement of data from a set of sensors [2]. For instance, [2] outlines a video surveillance application in which the sensors are cameras located at different locations over a geographical area. Another example arises in the area of network monitoring [3,4]. In this case routers produce streams of data pertaining to forwarded packets. More generally, stream processing can be seen as the execution of one of more "continuous queries" in the relational database sense of the term (e.g., a tree of join and select operators). Many authors have studied the execution of continuous queries on data streams [5,6].

In practice, the execution of the operators must be distributed over the network. In some cases the servers that produce the basic objects may not have the computational capability to apply all operators. Besides, objects must be combined across devices, thus requiring network communication. Sending all basic objects to a central compute server often proves unscalable due to network bottlenecks, or due to the central server not providing sufficient computational power. The alternative is to distribute the execution by mapping each node in the operator tree to one or more servers in the network, including servers that produce and update basic objects and/or servers that are only used for applying operators. One then talks of *in-network stream-processing*. Several in-network stream-processing systems have been developed [7,4]. These systems all face the same question: where should operators be mapped in the network?

In this paper we study the operator-mapping problem for *multiple concurrent in-network stream-processing applications*. The problem for a single application was studied in [8] for an ad-hoc objective function that trades off application delay and network bandwidth consumption. In a recent paper [9] we have studied a more general objective function, enforcing the constraint that the rate at which final results are produced, or *throughput*, is above a given threshold. This corresponds to a Quality of Service (QoS) requirement of the application that should be met while using as few resources as possible. In this paper we extend the work in [9] in two ways. First, we study a "non-constructive" scenario, i.e., we are given a set of compute and network elements, and we attempt to use as few resources as possible. Instead, in [9], we studied a "constructive" scenario in which resources could be purchased and the objective was to spend as little money as possible. Second, while in [9] we studied the case of a single application, in this paper we focus on multiple concurrent applications that contend for the servers, each with its own QoS requirement. Indeed, with several applications from several users running concurrently, it is more likely to share an existing set of resources for a common deployment, hence the call for the non-constructive scenario. Higher performance and reduced resource consumption is possible by reusing common sub-expression between operator trees when applications share basic objects [10]. We consider target platforms that are either fully homogeneous, or with a homogeneous network but heterogeneous servers, or fully heterogeneous. We formalize operator mapping problems for multiple in-network stream-processing applications and give their complexity; and we propose heuristics to solve the problems and evaluate them in simulation.

2 Framework

Application Model – We consider $\mathcal{K}$ applications, each needing to perform several operations organized as a binary tree (see Fig. 1). Operators are taken from the set $\mathcal{OP} = \{op_1, op_2, \dots\}$, and operations are initially performed on basic objects from the set $\mathcal{OB} = \{ob_1, ob_2, \dots\}$. These basic objects are made available and continuously updated at given locations in a distributed network. Operators higher in the tree rely on previously computed intermediate results, and they may also require to download basic objects periodically.

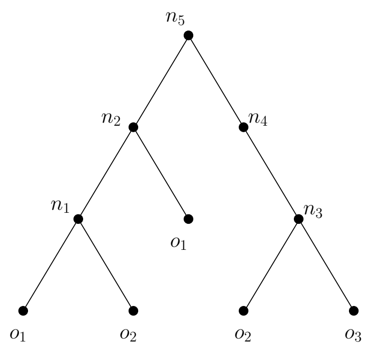


Fig. 1. Sample application structured as a binary tree of operators

For an operator op_p we define $objects(p)$ as the index set of the basic objects in $\mathcal{OB}$ that are needed for the computation of op_p, if any; and $operators(p)$ as the index set of operators in $\mathcal{OP}$ whose intermediate results are needed for the computation of op_p, if any. We have $|objects(p)| + |operators(p)| \leq 2$.

The tree structure of application k is defined with a set of labeled nodes. The i^{th} internal node in the tree of application k is denoted as $n_i^{(k)}$, its associated operator is denoted as $op(n_i^{(k)})$, and the set of basic objects required by this operator is denoted as $ob(n_i^{(k)})$. Node $n_1^{(k)}$ is the root node. Let $op_p = op(n_i^{(k)})$ be the operator associated to node $n_i^{(k)}$. Then node $n_i^{(k)}$ has $|operators(p)|$ child nodes.

The applications must be executed so that they produce final results, where each result is generated by executing the whole operator tree once, at a target rate. We call this rate the application *throughput*, $\rho^{(k)}$, specified as a QoS requirement for each application. Each operator in the tree of the k^{th} application must compute (intermediate) results at a rate at least as high as $\rho^{(k)}$. Conceptually, operator op_p executes two concurrent threads in steady-state. **(1)** It periodically downloads (or continuously stream) the most recent copies of the basic objects in $objects(p)$, if any. Basic object ob_j has size d_j (in bytes) and needs to be downloaded by the processors that use it for application k with frequency $f_j^{(k)}$. This consumes an amount of bandwidth of $rate_j^{(k)} = d_j \times f_j^{(k)}$ on each involved network link and network card. If a processor requires object ob_j for several applications with different update frequencies, it downloads the object only once at the maximum required frequency $rate_j = \max_k\{rate_j^{(k)}\}$. **(2)** It receives intermediate results computed by $operators(p)$, if any, and performs computation using basic objects it is continuously downloading and/or data received from other operators. The computation of operator op_p requires w_p operations, and produces an output of size δ_p.

Platform Model – The distributed network is a fully connected graph (i.e., a clique) interconnecting a set of processors $\mathcal{P}$. Operators are mapped onto these processors. Some processors also hold and update basic objects. Processor $P_u \in \mathcal{P}$ is interconnected to the network via a network card with maximum bandwidth B_u. The network link between two distinct processors P_u and P_v is bidirectional and has bandwidth $b_{u,v} (= b_{v,u})$, shared by communications in both directions. Processor $P_u \in \mathcal{P}$ has compute speed s_u. Processors that only provide basic objects and cannot compute are simply given compute speed 0. Resources operate under the full-overlap, bounded multi-port model [11]: Processor P_u can simultaneously compute, send, and receive data. With the "multi-port" assumption, each processor can send/receive data simultaneously on multiple network links. The "bounded" assumption enforces that the total transfer rate of data sent/received by processor P_u is bounded by its network card bandwidth, B_u.

Mapping Model and Constraints – The objective is to map internal nodes of application trees onto processors. If only one node is mapped to processor P_u, while P_u computes for the t-th final result it sends to its parent (if any) intermediate results for the $(t-1)$-th final result and it receives data from its children (if any) for computing the $(t+1)$-th final result. All three activities are concurrent. If several nodes are mapped to P_u the same overlap happens, but possibly on different result instances. A basic object can be duplicated, and thus available and updated at multiple processors. We assume that such duplication is achieved in some application-specific manner (e.g., via a distributed database that enforces sufficient data consistency). In this case, a processor can choose among multiple data sources for a basic object (or perform a local access if the basic object is available locally.)

We use an allocation function, a, to denote the mapping of the nodes onto the processors in $\mathcal{P}$: $a(k,i) = u$ if node $n_i^{(k)}$ is mapped to processor P_u. Conversely, $\bar{a}(u)$ is the index set of nodes mapped on P_u: $\bar{a}(u) = \{(k,i) \mid a(k,i) = u\}$. Also, we denote by $a_{op}(u)$ the index set of operators mapped on P_u: $a_{op}(u) = \{p \mid \exists (k,i) \in \bar{a}(u) \;\; op_p = op(n_i^{(k)})\}$. We introduce the following notations:

• $Ch(u) = \{(p,v,k)\}$ is the set of (operator, processor, application) tuples such that processor P_u needs to receive an intermediate result computed by operator op_p, which is mapped to processor P_v, at rate $\rho^{(k)}$; operators op_p are children of $a_{op}(u)$ in the operator tree.

• $Par(u) = \{(p,v,k)\}$ is the set of (operator, processor, application) tuples such that P_u needs to send to P_v an intermediate result computed by operator op_p at rate $\rho^{(k)}$; $p \in a_{op}(u)$ and the sending is done to the parents of op_p in the operator tree.

• $Do(u) = \{(j,v,k)\}$ is the set of (object, processor, application) tuples where P_u downloads object ob_j from processor P_v at rate $\rho^{(k)}$.

Given these notations, we can express constraints for the application throughput: each processor must compute and communicate fast enough to respect the prescribed throughput of each application with nodes allocated to it (Eq. 1). Note that each operator is computed only once at the maximum required throughput.

$$\forall P_u \in \mathcal{P} \quad \sum_{p \in a_{op}(u)} \left(\max_{(k,i) \in \bar{a}(u) \mid op(n_i^{(k)}) = op_p} \left(\rho^{(k)} \right) \frac{w_p}{s_u} \right) \leq 1 \, . \tag{1}$$

Communication occurs only when child and parent nodes are mapped on different processors. An operator computing for several applications may send/receive results to/from different processors. If the parent/child nodes corresponding to the different applications are mapped onto the same processor, the communication is done only once, at the most constrained throughput. In expressions below $v \neq u$ since we neglect intra-processor communications.

P_u must have enough bandwidth capacity to perform all its basic object downloads, to support downloads of the basic objects it may hold, and also to perform all communication with other processors, all at the required rates (Eq. 2). The first term corresponds to basic object downloads; the second term corresponds to download of basic objects from other processors; the third term corresponds to inter-node communications when a node is assigned to P_u and its parent node is assigned to another processor; and the last term corresponds to inter-node communications when a node is assigned to P_u and some of its children nodes are assigned to another processor.

$$\forall P_u \in \mathcal{P} \quad \sum_{\substack{(j,v,k)\\ \in Do(u)}} rate_j^{(k)} + \sum_{P_v \in \mathcal{P}} \sum_{\substack{(j,u,k)\\ \in Do(v)}} rate_j^{(k)} + \sum_{\substack{(p,v,k)\\ \in Ch(u)}} \delta_p \rho^{(k)} + \sum_{\substack{(p,v,k)\\ \in Par(u)}} \delta_p \rho^{(k)} \leq B_u \quad (2)$$

Finally, the link between processor P_u and processor P_v must have enough bandwidth capacity to support all possible communications between the nodes mapped on both processors, as well as the object downloads (Eq. 3).

$$\forall P_u, P_v \in \mathcal{P} \quad \sum_{\substack{(j,v,k)\\ \in Do(u)}} rate_j^{(k)} + \sum_{\substack{(j,u,k)\\ \in Do(v)}} rate_j^{(k)} + \sum_{\substack{(p,v,k)\\ \in Ch(u)}} \delta_p \rho^{(k)} + \sum_{\substack{(p,v,k)\\ \in Par(u)}} \delta_p \rho^{(k)} \leq b_{u,v} \quad (3)$$

Optimization Problems – The goal is to achieve a prescribed throughput for each application while minimizing a cost function. Several relevant problems can be envisioned. PROC-NB minimizes the number of used processors; PROC-POWER minimizes the compute capacity and/or the network card capacity of used processors (e.g., a linear function of both criteria); BW-SUM minimizes the sum of the used bandwidth capacities; and BW-MAX minimizes the maximum percentage of bandwidth used on all links. Different platform types may be considered depending on resource heterogeneity. We consider the fully homogeneous case ($s_u = s$, $B_u = B$ and $b_{u,v} = b$), which we term HOM. The case in which network links can have various bandwidths is termed HET.

3 Complexity

Problem PROC-NB is NP-complete in the strong sense even for a simple case: a HOM platform and a single application ($|\mathcal{K}| = 1$), that is structured as a left-deep tree [12], in which all operators take the same amount of time to compute and produce results of size 0, and in which all basic objects have the same size. We refer the reader to [9] for the proof. It turns out that the same proof holds for PROC-POWER on a HOM platform.

The BW-MAX problem is NP-hard because downloading objects with different rates on two processors is the same as the NP-hard 2-Partition problem [13]. Here is a sketch of the straightforward proof for a single application. Consider an application in which all operators produce zero-size results, and in which each basic object is used only by one operator. Consider three processors, with one of them holding all basic objects but unable to compute any operator. The two remaining processors are able to compute all the operators, and they are connected to the first one with identical network links. Such an instance can be easily constructed. The goal is to partition the set of operators in two subsets so that the bandwidth consumption on the two network links is as equal as possible. This is exactly the 2-Partition problem.

The BW-Sum problem can be reduced to the NP-hard Knapsack problem [13]. Here is a proof sketch for a single application. Consider the same application as for the proof of the NP-hardness of BW-Max above. Consider two identical processors, A and B, with A holding all basic objects. Not all operators can be executed on A and a subset of them need to be executed on B. Such an instance can be easily constructed. The problem is then to determine the subset of operators that should be executed on A. This subset should satisfy the constraint that the computational capacity of A is not exceeded, while maximizing the bandwidth cost of the basic objects associated to the operators in the subset. This is exactly the Knapsack problem.

All above problems can be solved via linear programming (see [14] for Integer Linear Program formulations). However, they cannot be solved in polynomial time (unless P=NP).

4 Heuristics

In this section we propose polynomial heuristics[1] for solving the Proc-Power problem when considering only the compute capacities of used processors. We propose 5 *heuristics* to map application nodes to processors. Each heuristic can use one of 4 generic *processor selection strategies* to select which processor a node should be mapped to. We consider two processor selection strategies, each with a *blocking* and a *non-blocking* version. *Blocking* means that once chosen for a given operator op_1, a processor cannot be used later for another operator op_2 unless op_2 is a relative (i.e., father or child) of op_1. *Non-blocking* heuristics impose no such restriction. We obtain four strategies *(S1) Select the fastest processor (blocking)*; *(S2) Select the processor with the fastest network card (blocking)*; *(S3) Select the fastest processor (non-blocking)*; and *(S4) Select the processor with the fastest network card.* Note that the processor and network card speeds used for the selection are computed while accounting for operators that may have already been mapped to servers.

All our heuristics attempt to re-use results from common operator sub-trees across applications. For this purpose they try to add additional communications as show in Fig. 2 on an example. We consider the following 5 heuristics:

• **(H1) Random** – H1 randomly picks the next node to map and attempts to reuse sub-trees across applications. If the node's operator has not already been mapped, possibly for another application, but the node's parent, H1 tries to map the node to the same processor. If unsuccessful, it makes similar attempts with the node's children. Otherwise if the node's operator has already been mapped somewhere else in the forest, H1 tries to add a communication from the already

[1] To ensure the reproducibility of our results, the code for all heuristics is available on the web [15].

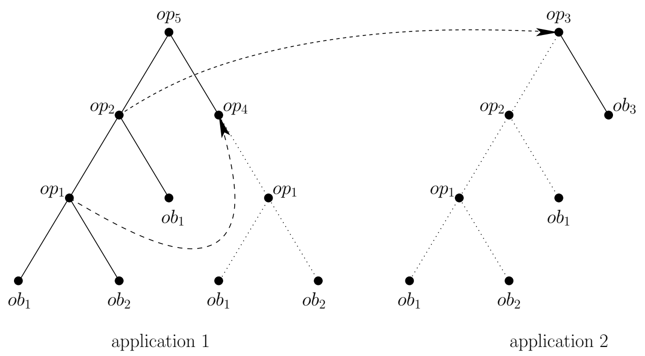


Fig. 2. Example for the reuse of nodes. op_1 is only computed once and its result is reused for the computation of op_2 and op_4. op_3 uses the result of op_2 in application 1 for its computation.

mapped operator to the father of the current node to reuse the common result. In this case, H1 marks the whole subtree (rooted at the operator) as mapped. Otherwise, H1 chooses a new processor according to the selected processor selection strategy. If unsuccessful, then H1 fails.

• **(H2) TopDownBFS** – H2 performs a breadth-first-search (BFS) traversal of all application trees, using an artificial node at which all application trees are rooted. For each node, H2 checks whether its operator has not been mapped yet and whether its father's has. In this case, H2 tries to map the operator on the same processor as its father, and in case of success continues the BFS traversal. If the node's operator has already been mapped, H2 tries to add a communication link between the mapped operator and the node's father: the mapped operator sends its result not only to its father but also to the node's father. If none of these two conditions holds, or if the mapping was not possible, H2 picks a processor according to the the processor selection strategy. If the mapping is successful, the BFS traversal continues, otherwise H2 fails.

• **(H3) TopDownDFS** – H4 uses the same mechanism as H2, but with a depth-first-search (DFS).

• **(H4) BottomUpBFS** – Like H2, H4 performs a BFS traversal of the application trees. For each node, H4 verifies whether it's operator has already been mapped. In this case a communication link is added (if possible), connecting the mapped operator and the node's father. If the operator is not yet mapped and if it has children, H4 tries to map the operator to one of its children's processors. If unsuccessful, or if the operator is at the bottom of a tree, H4 tries to map the operator onto a new processor chosen according to the processor selection strategy. If the mapping is successful, the traversal continues, otherwise H4 fails.

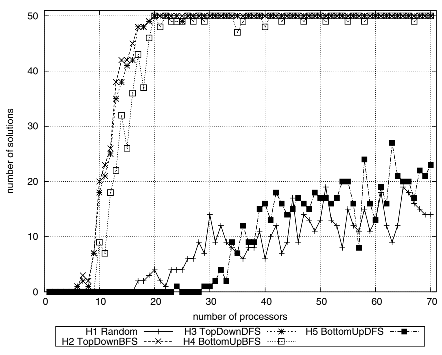
• **(H5) BottomUpDFS** – H5 is similar to H4, but uses a DFS traversal. This adds complexity as more cases need to be considered. For each node H5 checks if its operator has already been mapped and none of its children has. In this case H5 goes up in the tree until it reaches the last node n_1 such that there exists a node n_2 somewhere else in the forest whose operator is already mapped, and such that $op(n_1) = op(n_2)$. In this case H5 tries to add a communication between n_2 and the n_1's father to share a sub-tree. If the children have already been mapped H5 simply tries to map the operator to one of the children's processors. If this is not possible, or if the additional communication was not possible, or again if the operator has not been mapped anywhere in the forest, H5 tries to map the operator onto a new processor chosen according to the processor selection strategy. Otherwise H5 fails.

5 Experimental Results

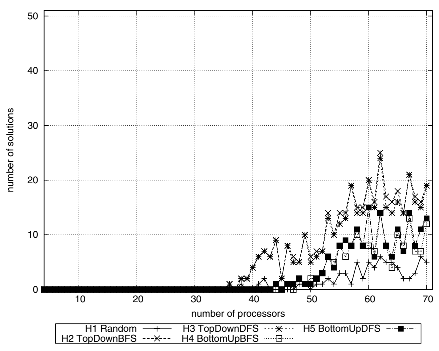
We have conducted several experiments to assess the performance of the different heuristics described in Section 4. In particular, we are interested in the impact of node reuse on the number of solutions found by the heuristics. The application trees are fixed to a size of at most 50 operators, and in general we consider 5 concurrent applications. The following parameters are chosen randomly: The basic objects (leaves in the tree) are chosen among 10 different types. The size d of each object type varies between 3MB and 13MB. The download frequencies of objects for each application, f, as well as the application throughput, ρ, are such that $0 < f \leq 1$ and $1 \leq \rho \leq 2$. The operands of operators are also chosen randomly. The computation amount w_i for an operator lies between 0.5MFlop/sec and 1.5MFlop/sec, and the output size of each operator δ_i varies between 0.5MB and 1.5MB. We dispose of 30 processors, equipped with a network card of bandwidth between 50MB and 180MB each. We use the same range for processor compute speed : 50MIPS to 180MIPS. Processors are interconnected via heterogeneous communication links, whose bandwidths are between 60MB/s and 100MB/s. The 10 different types of objects are randomly distributed over the processors, where objects are maximal twice available on processors. When deciding about basic object downloads, we first try to download from processors which are already used in the mapping (with minimal available bandwidth), before downloading from an unused processor. Execution time and communication time are scaled units, thus execution time is the ratio between computation amount and processor speed, while communication time is the ratio between object size (or output size) and link bandwidth.

We study the relative performance of each heuristic compared to the best solution found by any heuristic. This allows to compare the cost, in amount of resources used, of the different heuristics. The relative performance for heuristic h is obtained by: $\frac{1}{|runs|}\sum_{r=1}^{|runs|} a_h(r)$, where $a_h(r) = 0$ if heuristic h fails in run r and $a_h(r) = \frac{cost_{best}(r)}{cost_h(r)}$. $cost_{best}(r)$ is the best solution cost returned over all heuristics for run r, and $cost_h(r)$ is the cost in the solution returned by heuristic h. The number of runs is fixed to 50 in all experiments. The complete set of results is available on the web [15].

Summary of Experiments – We have performed different test series, varying the number of processors, the number of applications and the application size. Also we tested the impact of the Communication-to-Computation Ratio (CCR),



(a) Successful runs, strategy 3



(b) Successful runs without reuse, strategy 3

Fig. 3. Experiment: Increasing number of processors. Number of successful runs.

which is the ratio between the mean amount of communications and the mean amount of computations. Finally we were interested in the influence of application similarities on the heuristics' performance. Due to lack of space, we resume our experimental results, but a detailed description is available in [14].

Our results show that a random approach for multiple applications performs considerably bad. Not reusing results from common subtrees dramatically limits the success rate (see Fig. 3) and also the quality of the solution in terms of cost (relative performance). The TopDown approach turns out to be the best, and in most cases BFS traversal achieves the best result. The BottomUp approach is only competitive using a BFS traversal. The DFS traversal seems unable to reuse results efficiently (it often finds itself with no bandwidth left to perform necessary communications.) Furthermore we see a strong dependency of the processor selection strategy on solution quality. The blocking strategies outperform the non-blocking strategies when the CCR is large. Overall, H2 in combination with strategy S3 proves to be a solid combination.

6 Conclusion

We have studied the operator mapping problem of multiple concurrent in-network stream-processing applications onto a collection of heterogeneous processors. These applications come as a set of operator trees, that have to continuously download basic objects at different sites of the network and at the same time have to process this data to produce some final result. We have identified four relevant optimization problems. All are NP-hard but can be formalized as integer linear programs. Focusing on one of these optimization problems, we have designed several polynomial-time heuristics, which we have evaluated in simulation. Our experiments show the importance of node reuse across applications. Reusing nodes leads to an important number of additional solutions, and also the quality of the solutions improves considerably. We conclude that top-down traversal of the application trees is more efficient than bottom-up traversal.

As future work, we could develop heuristics for the other optimization problems defined in Section 2. We could also envision a more general cost function $w_{i,u}$ (time required to compute operator i onto processor u), in order to express even more heterogeneity. This would lead to the design of more sophisticated heuristics. Also, we believe it would be interesting to add a storage cost for objects downloaded onto processors, which could lead to new objective functions.

References

1. Badcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the Intl. Conf. on Very Large Data Bases, pp. 456–467 (2004)
2. Srivastava, U., Munagala, K., Widom, J.: Operator Placement for In-Network Stream Query Processing. In: Proceedings of the 24th ACM Intl. Conf. on Principles of Database Systems, pp. 250–258 (2005)
3. Cranor, C., Gao, Y., Johnson, T., Shkapenyuk, V., Spatscheck, O.: Gigascope: high-performance network monitoring with an SQL interface. In: Proc. of the ACM SIGMOD International Conference on Management of Data, pp. 623–633 (2002)
4. van Rennesse, R., Birman, K., Dumitriu, D., Vogels, W.: Scalable management and data mining using astrolabe. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 280–294. Springer, Heidelberg (2002)
5. Babu, S., Widom, J.: Continuous Queries over Data Streams. SIGMOD Record 30(3) (2001)
6. Plale, B., Schwan, K.: Dynamic Querying of Streaming Data with the dQUOB System. IEEE Trans. on Parallel and Distributed Systems 14(4), 422–432 (2003)
7. Chen, L., Reddy, K., Agrawal, G.: GATES: a grid-based middleware for processing distributed data streams. In: 13th IEEE International Symposium on High performance Distributed Computing, Proceedings, pp. 192–201 (2004)
8. Pietzuch, P., Leflie, J., Shneidman, J., Roussopoulos, M., Welsh, M., Seltzer, M.: Network-Aware Operator Placement for Stream-Processing Systems. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), pp. 49–60 (2006)
9. Benoit, A., Casanova, H., Rehn-Sonigo, V., Robert, Y.: Resource Allocation Strategies for Constructive In-Network Stream Processing. In: Proceedings of APDCM'09, the 11th Workshop on Advances in Parallel and Distributed Computational Models. IEEE, Los Alamitos (2009)
10. Pandit, V., Ji, H.: Efficient in-network evaluation of multiple queries. In: Robert, Y., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2006. LNCS, vol. 4297, pp. 205–216. Springer, Heidelberg (2006)
11. Hong, B., Prasanna, V.: Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. In: Intl. Parallel and Distributed Processing Symposium IPDPS 2004. IEEE Computer Society Press, Los Alamitos (2004)
12. Ioannidis, Y.E.: Query optimization. ACM Computing Surveys 28(1), 121–123 (1996)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability, a Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
14. Benoit, A., Casanova, H., Rehn-Sonigo, V., Robert, Y.: Resource Allocation for Multiple Concurrent In-Network Stream Processing Applications. Research Report 2009-07, LIP, ENS Lyon, France (February 2009)
15. Rehn-Sonigo, V.: Source Code for the Heuristics and diagrams of all experiments, `http://graal.ens-lyon.fr/~vsonigo/code/query-multiapp/`

Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of Their Functional Performance Models

Alexey Lastovetsky and Ravi Reddy

School of Computer Science and Informatics, University College Dublin,
Belfield Dublin 4, Ireland
{Alexey.Lastovetsky,Manumachu.Reddy}@ucd.ie

Abstract. The paper presents a new data partitioning algorithm for parallel computing on heterogeneous processors. Like traditional functional partitioning algorithms, the algorithm assumes that the speed of the processors is characterized by speed functions rather than speed constants. Unlike the traditional algorithms, it does not assume the speed functions to be given. Instead, it uses a computational kernel to estimate the speed functions of the processors for different problem sizes during its execution. This makes the algorithm distributed as its execution involves all the heterogeneous processors. The algorithm does not construct the complete speed function for each processor but rather builds and uses their partial estimates sufficient for optimal data distribution with a given accuracy. The low execution cost of this algorithm makes it ideal for employment in self-adaptable applications. Experiments with a parallel matrix multiplication application employing this algorithm are performed on a local heterogeneous computational cluster. The results show that the algorithm converges very fast and that its execution time is several orders of magnitude less than the total execution time of the application.

Keywords: distributed algorithms, data partitioning algorithms, functional performance models, heterogeneous platforms.

1 Introduction

Conventional data partitioning algorithms for parallel computing on heterogeneous processors [1-2] are based on a performance model, which represents the speed of a processor by a constant positive number, and computations are distributed amongst the processors such that their volume is proportional to this speed of the processor. The constant characterizing the performance of the processor is typically its relative speed demonstrated during the execution of a serial benchmark code solving locally the core computational task of some given size.

The traditional constant performance models (CPMs) proved to be accurate enough for heterogeneous distributed memory systems if partitioning of the problem results in a set of computational tasks that fit into the main memory of the assigned processors. But these models become less accurate in the presence of paging. The functional

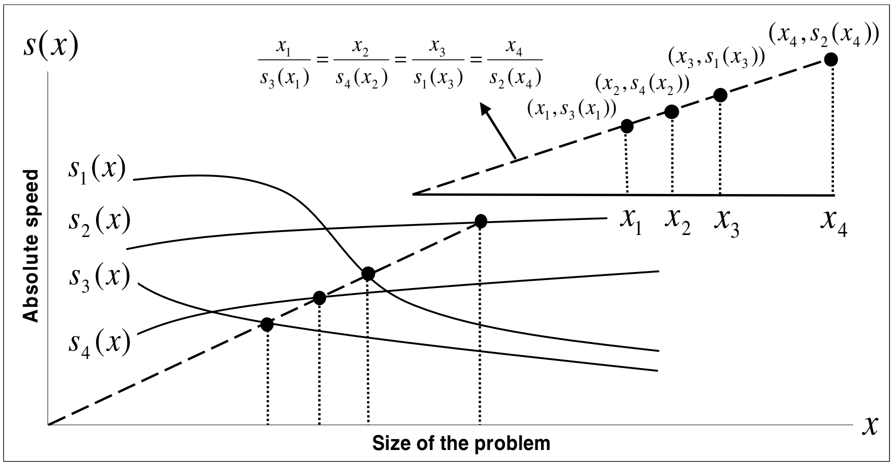


Fig. 1. Optimal data distribution showing the geometric proportionality of the number of chunks to the speed of the processor

performance model (FPM) of heterogeneous processors proposed and analyzed in [3] has proven to be more realistic than the CPMs because it integrates many important features of heterogeneous processors such as the processor heterogeneity, the heterogeneity of memory structure, and the effects of paging. The algorithms employing it therefore distribute the computations across the heterogeneous processors more accurately than the algorithms employing the CPMs. Under this model, the speed of each processor is represented by a continuous function of the size of the problem. This model is application centric because, generally speaking, different applications will characterize the speed of the processor by different functions.

The problem of distributing independent chunks of computations over a unidimensional arrangement of heterogeneous processors using this FPM has been studied in [3]. It can be formulated as follows: Given n independent chunks of computations, each of equal size (i.e., each requiring the same amount of work), how can we assign these chunks to p ($p<n$) physical processors P_1, P_2, ..., P_p with their respective full FPMs represented by speed functions $s_1(x)$, $s_2(x)$, ..., $s_p(x)$ so that the workload is best balanced? An algorithm solving this problem with a complexity of $O(p \times \log_2 n)$ is also proposed in [3]. This and other similar algorithms, which relax the restriction of bounded heterogeneity of the processors [4] and which are not sensitive to the shape of speed functions [5], are based on the observation that the optimal data distribution points $(x_1, s_1(x_1))$, $(x_2, s_2(x_2))$, …, $(x_p, s_p(x_p))$ lie on a straight line passing through the origin of the coordinate system and are the intersecting points of this line with the graphs of the speed functions of the processors. This is shown in Figure 1. These algorithms are used as building blocks in algorithms solving more complicated linear algebra kernels such as the dense factorizations [6].

The cost of experimentally building the full FPM of a processor, i.e., the FPM for the full range of problem sizes, is very high. This is due to several reasons. To start with, the accuracy of the FPM depends on the number of experimental points used to build it. The larger the number, the more accurate the FPM is. However, there is a cost associated with obtaining an experimental data point, which requires execution of a computational kernel for a specified problem size. This cost is especially high for problem sizes in the region of paging. Also, the number of experimental points required to build the full FPM increases remarkably as the number of parameters used to represent the problem size increases, as shown in the experimental results in this paper.

The problem of minimization of the cost of experimentally building the full FPM of the processor has been studied recently proposing a relatively efficient sub-optimal solution [7]. However, even if an ideal optimal procedure becomes available to build approximations of the FPM of heterogeneous processors, the fact remains that the cost of building the full FPM is too high to forbid the use of data partitioning algorithms, employing the full FPM, in self-adaptable applications.

The paper presents a new algorithm of data partitioning for parallel computing on heterogeneous processors. Like traditional functional partitioning algorithms, the algorithm assumes that the speed of the processors is characterized by speed functions rather than speed constants. Unlike the traditional algorithms, it does not assume the speed functions to be given. Instead, it uses a computational kernel to estimate the speed functions of the processors for different problem sizes during its execution. This makes the algorithm distributed as its execution involves all the heterogeneous processors. The algorithm does not construct the complete speed function for each processor but rather builds and uses their partial estimates sufficient for optimal data distribution. The proposed algorithm does not return a partitioning perfectly balancing the load of the processors but a partitioning balancing their load with a given accuracy.

Using experimental results for parallel matrix multiplication on a local heterogeneous computational cluster, we demonstrate that the execution time of the proposed distributed partitioning algorithm is several orders of magnitude less than the total execution time of the parallel application, thereby making it very suitable for employment in self-adaptable applications.

The rest of the paper is organized as follows. In Section 2, we present the contribution of this paper, which is the distributed iterative partitioning algorithm. This is followed by experimental results on a local heterogeneous computing cluster in Section 3. For the experiments, we use a parallel matrix multiplication application employing the data partitioning algorithm. Finally, we present numerical results demonstrating the efficiency of the distributed iterative partitioning algorithm.

2 Distributed Functional Partitioning Algorithm (DFPA)

The data partitioning problem that we are trying to solve can be formulated as follows:

- Given
  - A set of n independent units of computation each of equal size (i.e., each requiring the same amount of work);
  - A set of p ($p<n$) processors P_1, P_2, ..., P_p, whose speeds of processing x units, $s_i=s_i(x)$, can be obtained by measuring the execution time, $t_i(x)$, of a computational kernel, $s_i(x)=x/t_i(x)$,
  - ε, a required relative accuracy of the solution;

- Partition the set of computation units into p subsets so that

  There is one-to-one mapping between the partitions and the processors, and

  $\max_{1 \le i,j \le p}\left(\left|\frac{t_i(n_i)-t_j(n_j)}{t_i(n_i)}\right|\right) \le \varepsilon$, where n_i is the number of computation units allocated to processor P_i ($1 \le i \le p$).

Thus, the problem we study is to balance the load of heterogeneous processors with a given accuracy. The fundamental assumption, which makes efficient solution of this problem particularly difficult, is that the speeds of the processors are not known a priori. Therefore, if a partitioning algorithm needs the speed of processing of a given number of computation units by one or the other processor, it has to execute the corresponding number of units on this processor. Our solution to this problem is the following distributed data partitioning algorithm.

***Distributed Functional Partitioning Algorithm (DFPA)*:** The inputs to the algorithm are

- n, the number of computation units;
- p ($p<n$) processors $P_1, P_2, ..., P_p$;
- ε, the termination criterion.

The output d is an integer array of size p, the i-th element of which is the number of computation units allocated to processor i. The algorithm can be summarized as follows:

- **Initialization**:
  - All the p processors execute n/p computation units in parallel;
  - The execution times are gathered on processor P_1, $(t_1,\ldots,t_p) \leftarrow (t_1(n/p),\ldots,t_p(n/p))$;
  - If $\max_{1 \le i,j \le p}\left(\left|\frac{t_i(n/p)-t_j(n/p}{t_i(n/p)}\right|\right) \le \varepsilon$ then the even distribution of computations solves the problem and the algorithm stops;
  - Otherwise, processor P_1 calculates the absolute speeds of the processors, $s_i(n/p)=(n/p)/t_i$ for $1 \le i \le p$ and builds the first approximation of their FPMs in the form of constant models, $s_i(x) = s_i(n/p)$, as illustrated in Figure 2.

- **Iterating:** At each step,
  - Using the data partitioning algorithm [3], processor P_1 calculates a new distribution of computation units, $(d_1,\ldots,d_p)$, which will be optimal for the current approximations of the FPMs, and then sends a message to each processor P_i informing the latter of its new allocation of computation units, d_i ($1 \le i \le p$);

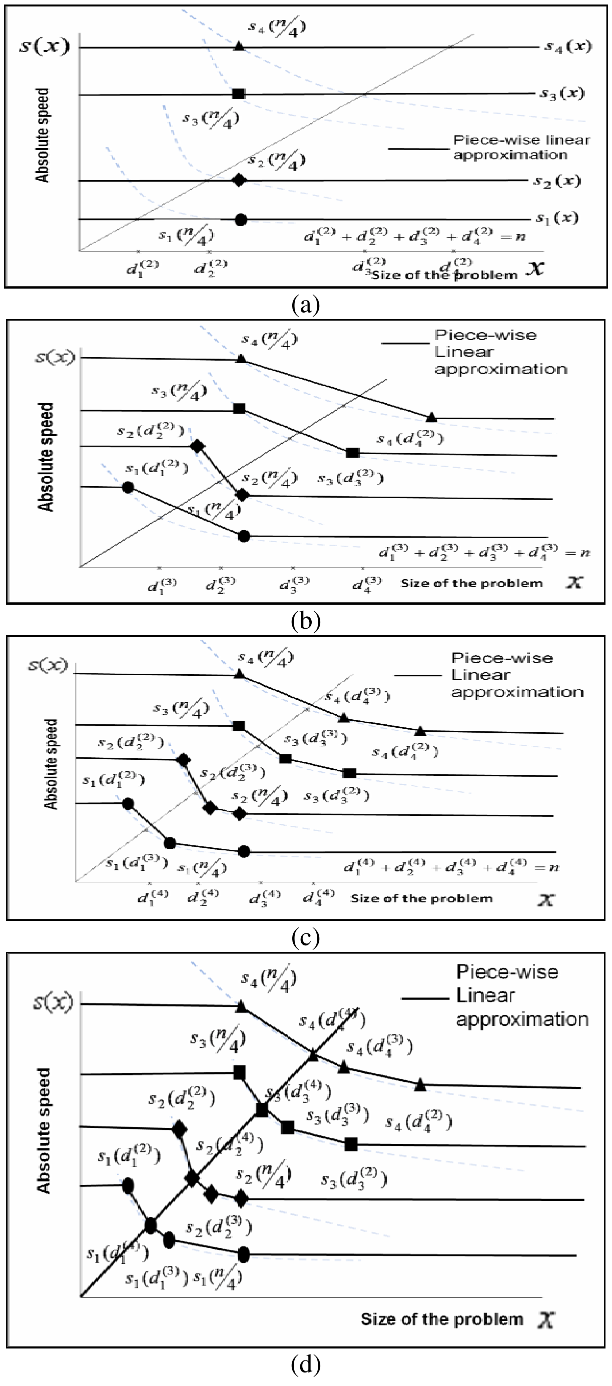

s(x)
Absolute speed
Piece-wise linear approximation
$s_4(x)$
$s_3(x)$
$s_2(x)$
$s_1(x)$
$s_4(n/4)$
$s_3(n/4)$
$s_2(n/4)$
$s_1(n/4)$
$d_1^{(2)}+d_2^{(2)}+d_3^{(2)}+d_4^{(2)}=n$
Size of the problem x
(a)
$d_1^{(3)}+d_2^{(3)}+d_3^{(3)}+d_4^{(3)}=n$
(b)
$d_1^{(4)}+d_2^{(4)}+d_3^{(4)}+d_4^{(4)}=n$
(c)
(d)

Fig. 2. Steps of the distributed functional partitioning algorithm (DFPA) illustrated using four heterogeneous processors. The dotted curves are real-life speed functions.

- Each processor P_i then executes d_i computation units in parallel with the other processors, $1 \le i \le p$;
- The execution times are gathered on processor P_1, $(t_1,\dots,t_p) \leftarrow (t_1(d_1),\dots,t_p(d_p))$;
- If $\max\limits_{1\le i,j\le p}\left(\left|\frac{t_i - t_j}{t_i}\right|\right) \le \varepsilon$, then the current distribution of computation units, $(d_1,\dots,d_p)$, solves the problem and the algorithm stops;
- Otherwise, processor P_1 calculates the absolute speeds, which the processors demonstrated for this distribution of computation units, $s_i(d_i) = \frac{d_i}{t_i}$ ($1 \le i \le p$), and uses these newly obtained points of the FPMs of processors P_i, $(d_i, s_i(d_i))$, to build their more accurate piecewise linear approximations (as illustrated in Figure 2). Namely, let $\{(d_i^{(j)}, s_i(d_i^{(j)}))\}_{j=1}^{m}$ ($d_i^{(1)} < \dots < d_i^{(m)}$) be the experimentally obtained points of $s_i(x)$ used to build its current piecewise linear approximation, then
  - If $d_i < d_i^{(1)}$, then the line segment $(0, s_i(d_i^{(1)})) \rightarrow (d_i^{(1)}, s_i(d_i^{(1)}))$ of this approximation will be replaced by two connected line segments $(0, s_i(d_{(i)})) \rightarrow (d_i, s_i(d_i))$ and $(d_i, s_i(d_i)) \rightarrow (d_i^{(1)}, s_i(d_i^{(1)}))$;
  - If $d_i > d_i^{(m)}$, then the line $(d_i^{(m)}, s_i(d_i^{(m)})) \rightarrow (\infty, s_i(d_i^{(m)}))$ of this approximation will be replaced by the line segment $(d_i^{(m)}, s_i(d_i^{(m)})) \rightarrow (d_i, s_i(d_i))$ and the line $(d_i, s_i(d_i)) \rightarrow (\infty, s_i(d_i))$;
  - If $d_i^{(k)} < d_i < d_i^{(k+1)}$, the line segment $(d_i^{(k)}, s_i(d_i^{(k)})) \rightarrow (d_i^{(k+1)}, s_i(d_i^{(k+1)}))$ will be replaced by two connected line segments $(d_i^{(k)}, s_i(d_i^{(k)})) \rightarrow (d_i, s_i(d_i))$ and $(d_i, s_i(d_i)) \rightarrow (d_i^{(k+1)}, s_i(d_i^{(k+1)}))$.
- Then, the algorithm proceeds to the next step.

Proposition. *Given the full FPMs of the processors P_1, P_2, ..., P_p satisfy the assumptions about their shape stated in [3], the DFPA algorithm always converges.*

Space limitations do not allow us to give the full formal proof of this proposition. In brief, its main points are as follows. First of all, by construction, the piecewise linear approximations of the full FPMs used in the algorithm will satisfy the same assumptions about their shape as the full FPMs themselves. Therefore, at each iteration step, application of algorithm [3] to the set of approximate FPMs will be successful and return the optimal solution for these approximate FPMs. Second, each next iteration step of the algorithm results in more accurate approximation of the segments of the full FPMs that contain the points of the optimal solution. Therefore, after a number of iterations, the approximations of the full FPMs will become accurate enough in order algorithm [3] to return a solution sufficiently close to the optimal one.

Figure 2 illustrates the operation of the DFPA algorithm using an example with four heterogeneous processors (P_1,P_2,P_3,P_4).

3 Experimental Results

We use a small heterogeneous local network of 16 different Linux processors (hcl01-hcl16) for the experiments. The specifications of the network are available at the URL http://hcl.ucd.ie/Hardware/Cluster+Specifications. The network is based on 2 Gbit Ethernet with a switch enabling parallel communications between the computers. The software used is MPICH-1.2.5 and ATLAS [8], which provides an optimized BLAS library.

Figure 3(a) shows the parallel matrix multiplication application. It implements matrix operation C=A×B, multiplying matrix A and matrix B, where A, B, and C are dense square matrices of size n×n matrix elements on a network of p heterogeneous processors. We use a 1D processor arrangement of size 3 for illustration purposes. Each element is a square matrix block of size b×b (the value of b used is 16). The matrices A and C must be horizontally sliced such that the height of the slice is

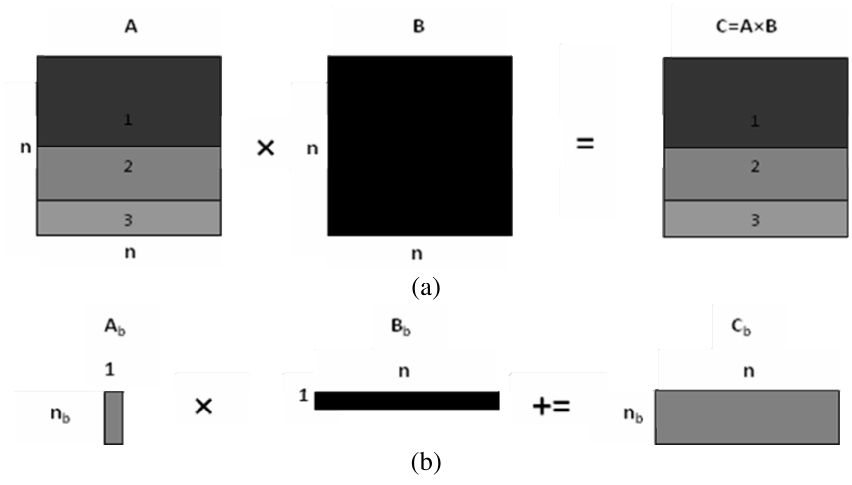


Fig. 3. (a) Matrix operation C=A×B on a network of three heterogeneous processors. Matrices A and C are horizontally sliced such that the height of the slice (n_b) is proportional to the speed of the processor. (b) The computational kernel (shown here for processor 2 for example) performs a matrix update of A_b of size n_b×1 and B_b of size 1×n to give a dense matrix C_b of size n_b×n. The matrix elements represent b×b matrix blocks.

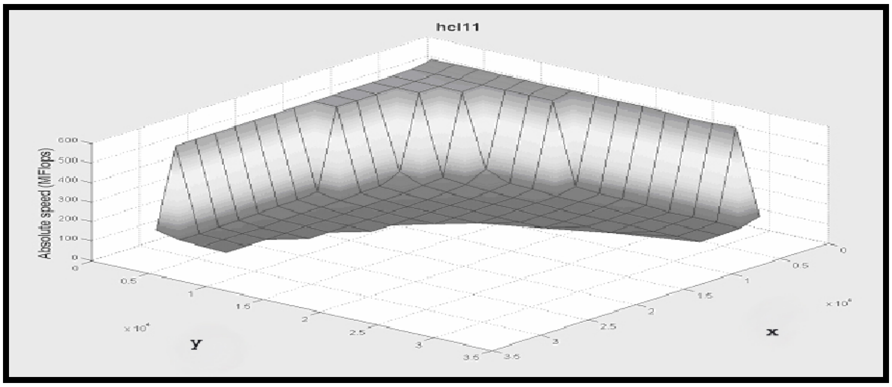
proportional to the speed of the processor owning the slice. All the processors contain all the elements of matrix *B*. We assume that only one process is configured to execute on a processor. We purposely choose an application with no communications because the goal of the experiments is not to show how to multiply matrices in parallel but to demonstrate the practical speed of convergence of the distributed partitioning algorithm. The results will not differ significantly for more complicated algorithms involving communications.

For this application, the core computational kernel performs a matrix update of a matrix C_b of size $n_b \times n$ using A_b of size $n_b \times 1$ and B_b of size $1 \times n$ as shown in Figure 3(b). Each element is a square matrix block of size $b \times b$. The size of the problem is represented by two parameters, n_b and n. The total number of matrix elements stored on each processor will be $(2 \times n_b \times n + n \times n)$. We use a combined computation unit, which is made up of one addition and one multiplication, to express the volume of computation. If n is large enough, the total number of computation units needed to solve this problem will be approximately equal to $n_b \times n$ (namely, multiplications of two $b \times b$ matrices). Therefore, the absolute speed of the processor exposed by the application when solving the problem of size (n_b, n) can be calculated as $n_b \times n$ divided by the execution time of the matrix update. This gives us a function, f: $\mathbf{N}^2 \rightarrow \mathbf{R}_+$, mapping problem sizes to speeds of the processor. The FPM of the processor is obtained by continuous extension of function f: $\mathbf{N}^2 \rightarrow \mathbf{R}_+$ to function g: $\mathbf{R}_+^2 \rightarrow \mathbf{R}_+$ $(\mathrm{f}(n,m)=\mathrm{g}(n,m)$ for any (n,m) from $\mathbf{N}^2)$. Figure 4(a) depicts this function for one of the processors, *hcl11*, used in experiments. Figure 4(b) shows the relative speed of two processors, *hcl09* and *hcl02*, calculated as the ratio of their absolute speeds. One can see that the relative speed varies significantly depending on the value of variables x and y (the variables represent n_b and n).

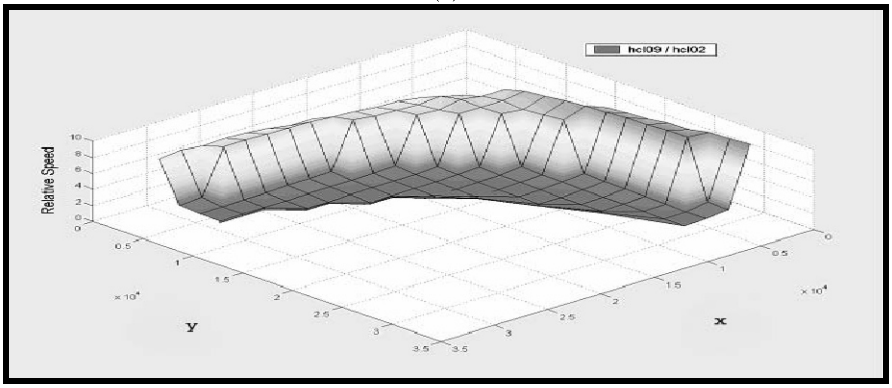
The heterogeneity of the network due to the heterogeneity of the processors is calculated as the ratio of the absolute speed of the fastest processor to the absolute speed of the slowest processor. For example, consider the benchmark code of a local DGEMM update of two matrices 2560×16 and 16×2560, the absolute speeds of the processors hcl01-hcl16 in million flop/s performing this update are {7696, 5196, 7852, 14418, 8000, 8173, 7288, 7396, 9037, 8987, 13661, 14194, 11182, 14410, 12008, 15257}. As one can see, hcl16 is the fastest processor and hcl02 is the slowest processor. The heterogeneity is therefore 3.

We compare the efficiency of the DFPA-based matrix multiplication application with the application based on the Full-Functional-Model Partitioning Algorithm (FFMPA). The difference between these applications is that the FFMPA-based one uses pre-built full FPMs of the processors for partitioning the matrices. More specifically, it uses the piecewise linear approximation of the full FPMs obtained with the GBBP procedure [7], which employs the same computational kernel as the DFPA-based application. Unlike the FFMPA-based application, the DFPA-based application does not need the FPMs of the processors as input. In all our experiments, the FFMPA returned the same data distribution as the DFPA.

Figure 5 shows the execution times of the sequential application and the parallel applications employing the FFPMA and DFPA and solving the same matrix multiplication problem. The sequential application uses optimized BLAS library (ATLAS) and is executed on the fastest processor (*hcl09*). The execution of the parallel matrix multiplication application consists of two parts. Firstly, all the processors execute the



(a)



(b)

Fig. 4. (a) The absolute speed of a processor ‘hcl11’ as a function of the size of the computational task of updating a dense *x*×*y* matrix. (b) The relative speed of two processors (‘hcl09’, ‘hcl02’) calculated as the ratio of their absolute speeds.

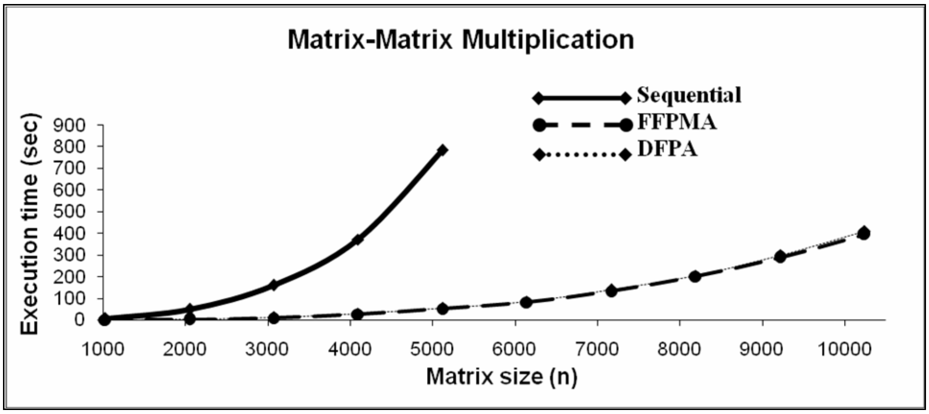


Fig. 5. Execution times of sequential and parallel applications with FFPMA and DFPA solving the same matrix multiplication problem

DFPA/FFPMA data partitioning algorithm to partition the matrices and then they perform the parallel matrix multiplication itself. For problem sizes (n>5120), the sequential application fails due to the problem size exceeding the memory limit of the processor. One can conclude that the parallel applications outperform the sequential application.

Table 1. Execution times of the parallel matrix multiplication application employing FFPMA and DFPA

Size of the matrix (n)	Number of iterations of DFPA	DFPA execution time (sec)	Execution time using DFPA (sec)	Execution time using FFPMA (sec)
1024	2	0.06	0.2	0.2
2048	2	0.09	2.2	1.9
3072	2	0.3	9.9	8.5
4096	5	2	28	25.3
5120	5	3	53.3	50.5
6144	5	3	84.4	80.7
7168	5	4	137.7	132.4
8192	5	5	204.3	199.7
9216	5	7	295.3	287.6
10240	5	11	405.9	393.3

Table 1 shows the execution times of the parallel matrix multiplication applications employing the FFPMA and the DFPA. The second column shows the number of iterations of DFPA. The third column shows the execution time of the DFPA. The fourth column shows the total execution time of the DFPA-based application. This includes the execution time of the DFPA. The fifth column shows the total execution time of the parallel application employing the FFPMA algorithm, which obviously does not include the time of construction of the FPMs of the processors.

One can see that the execution times of the parallel applications employing the FFPMA and DFPA differ only marginally. The difference is the execution time of the DFPA algorithm shown in the third column of Table 1. Most of it is spent in the partial estimation of the FPMs of the processors. It should be noted that the execution time of the parallel application employing the FFPMA does not take into consideration the time taken to build the full FPMs of the processors.

The execution time taken to build the full FPMs of the processors, which are used in the FFPMA-based application, is 425 seconds. The range of problem sizes, (n_b, n), used for building them satisfy the inequalities, $n_b \leq 10240$, $n \leq 10240$, and $n_b \leq n$. One can see that the execution time is significant compared to the DFPA execution times shown in the third column. The maximum number of experimental points used to build the full FPMs for this range is 60. This is compared to a maximum of 6 using DFPA (number of iterations plus one shown in column 2 of Table 1).

Thus, we can conclude that the DFPA converges very fast and its execution time is several orders of magnitude less than the execution time of the application. It is also efficient in terms of the number of experimental points.

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/IN.1/I2054.

References

[1] Kalinov, A., Lastovetsky, A.: Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers. Journal of Parallel and Distributed Computing 61(4), 520–535 (2001)

[2] Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Matrix Multiplication on Heterogeneous Platforms. IEEE Transactions on Parallel and Distributed Systems 12(10), 1033–1051 (2001)

[3] Lastovetsky, A., Reddy, R.: Data Partitioning with a Functional Performance Model of Heterogeneous Processors. International Journal of High Performance Computing Applications 21(1), 76–90 (2007)

[4] Lastovetsky, A., Reddy, R.: Data Partitioning for Multiprocessors with Memory Heterogeneity and Memory Constraints. Scientific Programming 13(2), 93–112 (2005)

[5] Lastovetsky, A., Reddy, R.: Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers. In: 17th International Parallel and Distributed Processing Symposium. IEEE Computer Society, Los Alamitos (2004)

[6] Lastovetsky, A., Reddy, R.: Data distribution for dense factorization on computers with memory heterogeneity. Parallel Computing 33(12), 757–779 (2007)

[7] Lastovetsky, A., Reddy, R., Higgins, R.: Building the Functional Performance Model of a Processor. In: 21st Annual ACM Symposium on Applied Computing, pp. 746–753. ACM Press, New York (2006)

[8] Automatically Tuned Linear Algebra Software (ATLAS), `http://math-atlas.sourceforge.net/`

An Efficient Weighted Bi-objective Scheduling Algorithm for Heterogeneous Systems[⋆]

Idalmis Milián Sardiña, Cristina Boeres, and Lúcia Maria de A. Drummond

Instituto de Computação, Universidade Federal Fluminense, Brazil
{isardina,boeres,lucia}@ic.uff.br

Abstract. This paper proposes the *Makespan and Reliability Cost Driven* (MRCD) heuristic, a static scheduling strategy for heterogeneous distributed systems that not only minimizes the makespan but also maximizes the reliability of the application. The scheduling decisions made by MRCD are guided by a weighted function that considers both objectives simultaneously instead of prioritizing only one of the objectives. This work also introduces a classification of the solutions produced by weighted bi-objective schedulers to aid users to tune the weighting function in order to generate an appropriate solution in accordance with their needs. In comparison with related work, MRCD produced schedules with makespans that were significantly better then those produced by other strategies at expense of an insignificant deterioration in reliability.

1 Introduction

Resources in large parallel and distributed systems may not be available for a long period of time. Therefore, when executing an application on such systems it is important to tackle reliability and fault-tolerance issues. Aiming an efficient execution of parallel applications on distributed heterogeneous system, this work specifies a schedule strategy that considers not only the minimization of the running time or *makespan*, but also maximizes the reliability of the application execution. The target application is represented by a directed acyclic graph (DAG), whose vertices are the tasks and edges are dependencies between them. A weighted bi-objective cost function that integrates both objectives guides the decisions to schedule the tasks of the parallel application on heterogeneous systems susceptible to failures.

In the literature, there is a variety of works that deal with this problem, which specifies in the cost function the two objectives in distinct ways. The work in [1] is a real time system oriented scheduling algorithm which orders the tasks of a parallel application by their deadlines. The cost function is a hierarchical one, since for each task, the subset of processors that maximizes the reliability is identified, and then the processor from this subset that minimizes the earliest start time is selected. As a consequence, the results favour the reliability rather than the execution time of the application. The authors in [2] derived the importance of

[⋆] This work is supported by research grants from CNPq and FAPERJ.

the product *failure rate* $\times$ *task execution time* for the case of scheduling independent tasks onto processors and proposed an extension of list scheduling heuristics, like HEFT [3], by taking into account reliability. In [4] a weighted bi-objective list scheduling algorithm (BSA) is proposed. After sorting the tasks in non-increasing order of their priority, the chosen processor is the one that minimizes a weighted integrated cost function. The objectives makespan and reliability are normalized by their maximum values and combined in the function. Nonetheless, [4] presents the use of only three solutions: the one that minimizes only the makespan; the one that weights equally both objectives; and the one that maximizes only the reliability. It must be pointed out that there are also works which manipulate multiple objective functions [5]. However, these approaches usually belong to a class of heuristics which are out of the scope of this paper.

The *Makespan and Reliability Cost Driven* (MRCD) proposed here is a list scheduling heuristic that takes into account the reliability and makespan based on [1]. However, instead of considering both objectives in a hierarchical fashion, the proposed cost function integrates both objectives simultaneously. The way in which the strategy was designed allows that the maximization of the reliability does not compromise the minimization of the makespan. The experimental analysis shows that MRCD produced schedules with very efficient performance when compared with other works from the literature. Besides, a classification of the solutions generated by weighted bi-objective schedules is presented as a way to aid the choice of a solution that better achieves the users' needs.

2 Scheduling Model for Distributed Systems with Faults

In this work, the *application model* is based on the class of parallel applications that can be represented by directed acyclic graphs (DAGs). A task graph is denoted by $G = (V, E, \varepsilon, \omega)$, where: the set of vertices V represents *tasks*; E, the precedence relation among them; $\varepsilon(v)$ is the amount of work associated with task $v \in V$; and $\omega(u, v)$ is the weight associated with the edge $(u, v) \in E$, representing the amount of data units transmitted from task u to v.

The *architectural model* specifies the main features of the target architecture. Given the set $P = \{p_0, p_1, \ldots, p_{m-1}\}$ of available heterogeneous processors, the computational slowdown index $csi(p_j)$ is associated to each $p_j \in P$, which is inversely proportional to the computational power of p_j, as identified by [6]. The communication delay index $cdi_{i,j}$ estimates the latency cost associated with each communication link (p_i, p_j). Therefore, the execution time of a task v on a processor p_j is $et(v, p_j) = \varepsilon(v) \times csi(p_j)$. The communication time to send the amount $\omega(u, v)$ of data between the adjacent tasks u and v allocated to distinct processors p_u and p_v is given by $ct(u, v) = \omega(u, v) \times cdi(p_u, p_v)$. Note that $cdi(p_u, p_v) = 0$ if $p_u = p_v$. Upon the definition of these costs, a schedule algorithm may need to calculate the earliest time that a task v can start its execution on processor p_j, which is $EST(v, p_j) = \max_{u \in pred(v)}\{EST(u, p_u) + et(u, p_u) + ct(u, v)\}$ and depends on the schedule of v's predecessor tasks and the availability of the processor. The earliest finish time of task v on processor p_j is then $EFT(v, p_j) =$

$EST(v, p_j) + et(v, p_j)$. A schedule Sch of a DAG G specifies, for each $v \in V$, the processor and the time in which v must be completed. The schedule length or *makespan* of Sch is defined as $\mathcal{M}(Sch) = \max_{\forall v \in V}\{EFT(v, p_v)\}$.

The reliability of a system is based on the probability in which resources of the system execute tasks without any failure [1]. In this work, only processor failures are considered, which are statistically independent and follow a Poisson Law with a failure rate of $FR(p_j)\ \forall p_j \in P$ [7]. This rate represents the number of processor failures per unit of time that can occur. It is assumed here that the communication links between processors are reliable, based on the assumption that communication protocols are able to tolerate their failures [7].The task reliability cost associated with the execution of v in p_j is then $RC(v, p_j) = FR(p_j) \times et(v, p_j)$, which should be minimized. Given the set of tasks allocated to p_j, $V_{p_j} \subset V$, the processor reliability cost is then $RC_p(p_j) = \sum_{\forall v \in V_{p_j}} RC(v, p_j)$. As a consequence, the system reliability cost associated with the execution of G on P in accordance with a schedule Sch is $RC_s(G, P, Sch) = \sum_{\forall p_j \in P} RC_p(p_j)$. Finally, as in [1], the total reliability of the scheduled G in P is $R_T = e^{-RC_s(G,P,Sch)}$, which is the probability that the application G can run successfully on the set P under the schedule Sch during its execution. As a matter of fact, R_T should be maximized and also, by minimizing the task reliability cost $RC(v, p_j)$, the total reliability R_T becomes close to one.

3 A Weighted Bi-objective Scheduling Strategy

The MRCD scheduling algorithm follows the traditional *list scheduling* framework: firstly, during the *ordering phase*, a priority is associated with the tasks. Then, during the *scheduling* phase, each task with the highest priority is allocated to the processor that minimizes a cost function. This work defines a weighted bi-objective function to guide the scheduling process, such that not only the makespan of the application is minimized but also the total reliability of the application scheduled in P is maximized. The specification of proper weights to the objectives can be however, troublesome. On an attempt to provide information to aid this specification, a metric is also generated by MRCD, which is the average difference between the two objectives.

The bottom level $blevel(v)$ is the priority assigned to the tasks in V as in [3]. During the *scheduling* phase, the algorithm seeks for the processor that optimizes the cost function. For doing so, MRCD also implements the *tasks insertion* procedure from [3]. Let v be the next task to be scheduled with the highest $blevel(v)$. The chosen processor p_v is the one that minimizes the cost function $f(v, p_v) = \min_{p_j \in P}\{(1 - w)EFT(v, p_j) + wRC(v, p_j)\}$. When $w = 0$, MRCD becomes similar to HEFT [3], in which the objective is only the minimization of the makespan. When $w = 1$, MRCD only considers the reliability.

A closer look to $f(v, p_j)$ can take one to note that the values of the objectives are not comparable and aggregating them is not straightforward. In Therefore, it is necessary to normalize both objectives $EFT(v, p_j)$ and $RC(v, p_j)$ for each task in V over the set of processors P. The normalization procedure transforms all the objectives so that they share the same minimum and maximum values 0 and 100. Then, the linear operator applied to the i^{th} objective is $O_n^i = norm(0, 100, O_i^{min}, O_i^{max}) = 100 \times \frac{O_i - O_i^{min}}{O_i^{max} - O_i^{min}}$, where O_i^{min} and O_i^{max} are the minimum and maximum values of the i^{th} objective, respectively. In this work, O_n^i can be either the normalized values $EFT_n(v, p_j)$ or $RC_n(v, p_j)$.

Algorithm 1. *MRCD(G,P,w)*

```
$V_{ord} = \langle v_0, v_1, \ldots, v_{n-1} \rangle / blevel(v_i) \leq blevel(v_{i+1}), i = 0, \ldots, n-2$;
for $i = 0, \ldots, n-1$
    $F = \infty$;
    $\forall p_j \in P$
        Calculate $EST(v_i, p_j)$ using $taskInsertion(v_i, p_j)$;
        $EFT(v_i, p_j) = EST(v_i, p_j) + et(v_i, p_j)$;
        $RC(v_i, p_j) = FR(p_j) \times et(v_i, p_j)$;
    $EFT_{min} = \min_{p_j \in P}\{EFT(v_i, p_j)\}$; $EFT_{max} = \max_{p_j \in P}\{EFT(v_i, p_j)\}$;
    $RC_{min} = \min_{p_j \in P}\{RC(v_i, p_j)\}$;
    $RC_{max} = \max_{p_j \in P}\{RC(v_i, p_j)\}$;
    $\forall p_j \in P$
        $EFT_n(v_i, p_j) = norm(0, 100, EFT_{min}, EFT_{max}, EFT(v_i, p_j))$;
        $RC_n(v_i, p_j) = norm(0, 100, EFT_{min}, RC_{max}, RC(v_i, p_j))$;
        $f(v_i, p_j) = (1 - w) \times EFT_n(v_i, p_j) + w \times RC_n(v_i, p_j)$;
        if $(f(v_i, p_j) < F)$
            $F = f(v_i, p_j)$; $p_{v_i} = p_j$;
    if $(\mathcal{M}(Sch) < EFT(v_i, p_{v_i}))$ $\mathcal{M}(Sch) = EFT(v_i, p_{v_i})$;
    $RC_s = RC_s + RC(v_i, p_{v_i})$;
    $Sch = Sch \cup \langle v_i, p_{v_i}, EST(v_i, p_{v_i}) \rangle$;
$R_T = e^{-RC_s}$; $D(Sch) = \frac{\sum_{v_i \in V} RC_n(v_i, p_{v_i}) - EFT_n(v_i, p_{v_i})}{n}$;
```

Algorithm 1 shows the steps of the MRCD Algorithm for a given $G = (V, E, \epsilon, \omega)$ and set of processors P. The *ordering* phase of MRCD generates the list V_{ord} of tasks in non-decreasing order of $blevel(v)$ (line 1). The *scheduling* phase is executed for each $v_i \in V_{ord}$ from lines 2 to 19, where in lines 4 to 7, the earliest finishing time $EFT(v_i, p_j)$ and the minimal task reliability $RC(v_i, p_j)$ are calculated, for each $p_j \in P$. In lines 8 to 10, both the minimal and maximal values for $EFT(v_i, p_j)$ and $RC(v_i, p_j)$ over all $p_j \in P$ are collected and then applied to the normalization procedure. The normalized values of $EFT_n(v_i, p_j)$ and $RC_n(v_i, p_j)$ are then applied to the cost function $f(v_i, p_j)$ where the processor which minimizes it is identified (lines 15 and 16). MRCD generates the final schedule *Sch*, its makespan $\mathcal{M}(Sch)$ and the total reliability cost R_T, as seen in lines 17, 19 and 20, respectively. Furthermore, with the normalized values $EFT_n(v_i, p_{v_i})$ and $RC_n(v_i, p_{v_i})$, for each v_i in its best processor p_{v_i}, their difference portrays the imbalance degree between both objectives. The average difference or imbalance degree between the objectives $D(Sch)$ of the resulting schedule Sch is given by $D(Sch) = \frac{\sum_{v_i \in V} RC_n(v_i,p_{v_i}) - EFT_n(v_i,p_{v_i})}{n}$, as seen in line 20. The sign of $D(Sch)$ shows which objective most contributed to the specification of the produced schedule. If negative, the reliability cost contributed more, otherwise, the makespan dominates. In this way, if its module $|D(Sch)|$ is close to zero, the imbalance degree is practically negligible and an equally contribution of the objectives are held.

3.1 A Classification of the Weights Applied to the Cost Function

When many and probably conflicting objectives are optimized simultaneously, there is no longer a single optimal solution but rather a whole set of possible solutions of equivalent quality. In the problem tackled in this work, the two objectives might be conflicting: while a processor can finish the execution of an application task quickly, a high failure rate can be assigned to it depending on the reliability model. Therefore, there is no single optimum solution to be found and actually a variety of solutions can be optimal in some sense. This work provides the users the option to assess the trade-offs between the objectives with additional information about this set of solutions.

Let S be the set of all feasible solutions for the bi-objective problem tackled here. Let $S_k = \langle Sch_k, \mathcal{M}(Sch_k), R_T(Sch_k), D(Sch_k)\rangle$ be one solution of the bi-objective problem and the associated additional information to be analysed. A solution $S_k \in S$ dominates another solution S_q if the following conditions are satisfied: (i) S_q is not better than S_k in both objectives, i.e. $\mathcal{M}(Sch_k) \leq \mathcal{M}(Sch_q)$ and $R_T(Sch_k) \geq R_T(Sch_q)$; (ii) S_k is absolutelly better than S_q in at least one of the objectives, i.e. $\mathcal{M}(Sch_k) < \mathcal{M}(Sch_q)$ or $R_T(Sch_k) > R_T(Sch_q)$. The solution S_k is dominant and S_q is dominated by S_k. If S_k does not dominate S_q and vice-versa, the solutions are incomparable [5]. A variety of feasible solutions can be produced by MRCD for the same input DAG G and target system if different w values are given to the algorithm. Let W be the set of values associated with w of the cost function in MRCD. In accordance with the dominance condition (i) and (ii) described above, the set of all solution in S that are dominants and incomparable is denoted as S' (non-dominated). Note that the solutions in S' are not dominated by any other solution in S. The concept of dominance can be sometimes too weak for applications, in cases where one objective can be improved significantly at a cost of a small deterioration of the other. This concept does not necessarily tell which solutions to chose, but rather which solutions to avoid [5].

A classification is then proposed, having the knowledge of the solutions in S'. The solution $S_e \in S'$ that offers an equilibrium between the objectives are those with the smallest value of $|D(Sch_e|$. The set of solutions S_{R_T} is compounded of those $S_k \in S'$ with the smallest $D(Sch_k)$, and consequently are the ones with the highest total reliability $R_T(Sch_k)$. On the other hand, the set of solutions denoted by $S_{\mathcal{M}}$ contains the $S_k \in S'$ with the highest $D(Sch_k)$, i.e. are those with the minimum makespan $\mathcal{M}(Sch_k)$.

4 Experimental Results

The proposed scheduling algorithm MRCD was compared with other scheduling strategies from the literature, which were divided into two classes: the first one produces a unique solution, denoted as Class 1, encompasses the heuristics HEFT [3], RCDMod (based on [1]) and RHEFT (based on [2]); and Class 2, containing BSAMod based on [4] that can also produce several solutions. BSAMod emplows a cost function that although seems similar to MRCD produces different results, added to the fact that a distinct normalization procedure is used. In order to have a fair comparison with MRCD, some of these heuristics were implemented with small changes, as will be describe next.

In Class 1, since HEFT minimizes only the makespan of the resulting schedule, to fairly compare with MRCD, this work derived R_T based on the final schedule produced by HEFT. The work [1] also gives a solution for the bi-objective scheduling problem, but uses a hierarchical cost function. Differently from the original proposal, RCDMod was implemented with no time constraints (deadlines) and the employed cost function used the tasks completion times instead of their start times. Finally, RHEFT implements the list scheduling approach of HEFT, but considers both objectives in its cost function as the multiplication of the execution time and failure rate as in [2]. In Class 2, BSAMod also tackles the bi-objective problem by using the function $f(v,p) = \sqrt{w\left(\frac{EFT(v,p_j)}{\max_{p\in P}\{EFT(v,p)\}}\right)^2 + (1-w)\left(\frac{RC(v,p_j)}{\max_{p\in P}\{RC(v,p)\}}\right)^2}$, as proposed in [4], but sorts tasks by $blevels(v)$ and executes the task insertion procedure.

All heuristics were executed over synthetic applications represented by three classes of DAGs: one that represents the Gaussian Elimination, denoted by G_n; the diamond DAG Di_n, which represents, for example, matrix multiplication; and random generated DAGs R_n, graphs with irregular topologies. In all the cases, n denotes the number of tasks of the DAG. In all graphs, a unit cost was associated with the communications between tasks. The same does not hold for the tasks weights. In G_n, each $v \in V$ has a distinct computation weight, while in both R_n and Di_n, the weight $\epsilon(v) = 50$.

The simulated distributed system was composed of m processors divided into three groups, denoted as P_0, P_1 and P_2, each one with $m/3$ processors. The processor failure rate, $FR(p_i)$, was uniformly generated in $[10^{-5}, 3.3 \times 10^{-5}]$, $\forall p_i \in P_0$; $[3.4 \times 10^{-5}, 6.6 \times 10^{-5}]$, for $p_i \in P_1$; $[6.7 \times 10^{-5}, 10^{-4}]$, for $p_i \in P_2$, as proposed in [1]. Concerning the computational slowdown index, the adopted values were $csi(p_i) = 73$, for $p_i \in P_0$; $csi(p_i) = 53$, for $p_i \in P_1$; $csi(p_i) = 33$, for $p_i \in P_2$, which were obtained from [6]. The failure rate is per second and the makespan value in seconds.

An illustration of the proposed scheduling algorithm and the classification of w values given by MRCD considering G_{702} and the architectural scenario with 24 processors is shown in Table 1 where: w is the weight applied to the cost function $f(v, p_j)$; $\mathcal{M}$ is the makespan of the produced schedule; R_T is the total reliability; and the associated average difference D. The columns in bold show S', the non-dominated solutions. Among them, three solutions are particularly interesting under the classification proposed in Section 3.1: $S_{\mathcal{M}}$, S_{R_T} and S_e for $w = 0.4$, $w = 0.9$ and $w = 0.7$, respectively. From the definition of D and the normalized values RC_n and EFT_n in Algorithm 1, for high values of w and consequently, low values of D, MRCD tends to maximize reliability, otherwise, the makespan is minimized.

Table 1. Solution provided by MRCD for G_{702}

w	0.1	0.2	0.3	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
$\mathcal{M}$	902.8	902.8	902.8	**902.8**	**906.3**	**1266.4**	**3463.3**	**7072.2**	**12687.4**
RT	0.44084	0.44768	0.45400	**0.45539**	**0.46016**	**0.48362**	**0.53184**	**0.59064**	**0.62604**
D	49.1	45.7	42.8	**41.0**	**38.7**	**27.8**	**-4.8**	**-36.3**	**-57.3**

4.1 Comparing MRCD with Other Heuristics

The first set of experiments compares the results of MRCD with the heuristics in Class 1. Table 2 presents the pair makespan and the system reliability $(\mathcal{M}, R_T)$, and also average difference D for each DAG, considering the three following comparisons. The first comparison C1 gives the results of HEFT and MRCD, C2 shows the results of RCDMod and MRCD, and C3 compares RHEFT and MRCD. In each case, the MRCD solutions selected were: $S_{\mathcal{M}}$ from S' in C1 for a fair comparison with HEFT; $S_{R_T} \in S'$ in the case of C2, since RCDMod gives higher priority to the processors that maximizes reliability; and the dominant solution or in case of MRCD solutions do not dominate the RHEFT solution, the MRCD solution closest to the RHEFT one is shown. A solution S_i is the closest one to another solution S_j if it presents the smallest Euclidian distance from it considering normalized values.

It can be observed that with the increase of n, the makespan also grows and the reliability decreases. As the number of tasks executed on a fixed number of processors grows, the processors loads also increase and consequently their reliability costs. Table 2 also shows that the solutions in C1 have worse reliability than the results in C2 and C3, but the makespan are amazingly smaller. It happened because in the employed environment, the fastest processors were also the most susceptible to failures, and the chosen values for w gave priority to the minimization of the makespan. Most interestingly, MRCD presented better solutions than HEFT in all graphs. By choosing MRCD solutions with the greatest D values (the $S_{\mathcal{M}}$ solution), the algorithm showed to be capable of producing solutions with makespans as good as (or better than) those of HEFT, but with better reliability.

In C2, RCDMod presented slightly better reliability than the results S_{R_T} for each DAG in Table 2. Remark that such heuristic employs a hierarchical cost function that prioritizes the reliability only, and although the MRCD solutions also consider reliability, the makespan was not neglected since the cost function integrates both objectives. However, it is important to note that the makespan of the RCDMod are almost the double of those presented by MRCD. Finally in C3, MRCD solutions dominated RHEFT solutions in many graphs. RHEFT employs

Table 2. Results for Gauss, Random and Diamond DAGs on $m = 24$ processors in Class 1

	C1			C2			C3		
DAG_N	HEFT	MRCD	D	RCDMod	MRCD	D	RHEFT	MRCD	D
G_{152}	(190.0, 0.92071)	**(190.0, 0.93294)**	30.3	(2406.0, 0.95761)	(1283.3, 0.95546)	-58.7	(694.9, 0.94573)	(449.4, 0.94287)	-13.2
G_{252}	(318.7, 0.83511)	**(318.7, 0.85879)**	31.0	(5201.9, 0.91061)	(2715.6, 0.90609)	-58.1	(1309.6, 0.88130)	(856.3, 0.87844)	-7.6
G_{377}	(480.4, 0.71847)	**(480.4, 0.75140)**	32.7	(9603.8, 0.84124)	(4981.5, 0.83350)	-57.2	(2204.6, 0.78693)	(1494.8, 0.78560)	-6.3
G_{527}	(675.1, 0.58537)	**(675.1, 0.61228)**	35.6	(15976.7, 0.75007)	(8211.0, 0.73851)	-57.1	(3560.9, 0.66798)	(2273.0, 0.66578)	-4.4
G_{702}	(902.8, 0.44064)	**(902.8, 0.45539)**	41.0	(24685.6, 0.64124)	(12687.4, 0.62604)	-57.3	(5258.9, 0.52865)	**(3463.3, 0.53184)**	-4.8
R_{80}	(330.0, 0.82648)	**(330.0, 0.84159)**	27.7	(5840.0, 0.90021)	(3066.0, 0.89523)	-58.7	(1022.0, 0.85271)	**(949.0, 0.86478)**	-13.2
R_{98}	(330.0, 0.79053)	**(330.0, 0.80765)**	29.9	(7154.0, 0.87917)	(3650.0, 0.87303)	-58.7	(1460.0, 0.83060)	**(1314.0, 0.84035)**	-10.6
R_{152}	(396.0, 0.69599)	**(396.0, 0.69907)**	42.0	(11096.0, 0.81895)	(5694.0, 0.81015)	-56.9	(1679.0, 0.73349)	**(1533.0, 0.75338)**	-5.95
R_{256}	(528.01, 0.54215)	**(528.00, 0.54215)**	38.2	(18688.0, 0.71434)	(10512.0, 0.70276)	-60.0	(1606.01, 0.56659)	**(1314.0, 0.58304)**	12.8
R_{364}	(759.0, 0.41596)	**(759.0, 0.41804)**	49.2	(26572.0, 0.61983)	(13505.0, 0.60385)	-54.5	(2993.0, 0.45685)	**(1825.0, 0.46744)**	15.3
Di_{81}	(580.9, 0.83000)	**(580.9, 0.83218)**	43.6	(5913.0, 0.89903)	(3066.0, 0.89393)	-59.7	(3285.0, 0.89249)	**(3066.0, 0.89393)**	-59.7
Di_{100}	(660.0, 0.79495)	**(660.0, 0.79665)**	43.9	(7300.0, 0.87686)	(3723.0, 0.87061)	-60.5	(4599.0, 0.87214)	(3723.0, 0.87061)	-60.5
Di_{144}	(825.0, 0.71535)	**(825.0, 0.71903)**	44.7	(10512.0, 0.82760)	(5329.0, 0.81907)	-61.0	(6935.0, 0.82170)	(5329.0 0.81907)	-61.0
Di_{256}	(1155.0, 0.54076)	**(1155.0, 0.54940)**	46.6	(18688.0, 0.71434)	(9417.0, 0.70122)	-62.5	(12994.0, 0.70626)	(9417.0, 0.70122)	-62.5
Di_{361}	(1452.0, 0.41585)	**(1452.0, 0.42095)**	49.5	(26353.0, 0.62228)	(13286.0, 0.60623)	-62.7	(19126.0, 0.61335)	(13286.0, 0.60623)	-62.7

a cost function that also integrates both objectives but in a distinct manner than from MRCD. In C3, MRCD presented makespans which are almost half of those produced by RHEFT, while the reliabilities were very close.

The second set of experiments was performed with a varying number of processors, on the DAGs: G_{1034}, R_{546} and Di_{529}, as seen in Table 3. With an increasing number of processors, the makespan diminishes while reliability increases, since the scheduling strategies can allocate tasks on more appropriate processors concerning failure rates and computational slowdown indexes. In relation with the comparisons C1, C2 and C3, the same behaviour previously described is also detected.

9 The next sets of experiments were performed considering BSAMod of Class 2. The set S was generated by executing both MRCD and BSAMod with $W = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ for 24 processors, initially. A varying number of tasks and processors were also considered in these experiments. Table 4 shows the percentage difference of both makespan and reliability for each w value. The line $\mathcal{M}$ presents the percentage improvement on the makespan produced by MRCD over BSAMod and in line R_T, the percentage improvements on the reliability of BSAMod over MRCD. Note that an outstanding improvement on the makespan is produced by MRCD while the reliability remains almost the

Table 3. Comparison in Class 1 for G_{1034}, R_{546} and Di_{529} with a varying number m of processors

		C1			C2			C3		
DAG_N	m	HEFT	MRCD	D	RCDMod	MRCD	D	RHEFT	MRCD	D
G_{1034}	24	(1504.1, 0.86253)	**(1499.8, 0.86628)**	41.3	(44389.8, 0.92320	(22758.4, 0.91922)	-55.8	(8743.9, 0.88970)	**(5309.0, 0.89017)**	2.1
	45	(1335.8, 0.84661)	**(1335.8, 0.88030)**	35.2	(44389.8, 0.93558)	(23805.3, 0.92982)	-51.5	(9291.4, 0.91737)	**(4623.8, 0.91816)**	-11.0
	66	(1335.8, 0.85161)	**(1335.8, 0.89391)**	31.5	(22229.9, 0.95234)	(9640.3, 0.94698)	-49.8	(5673.5, 0.93053)	**(4940.6, 0.93929)**	-29.3
	87	(1335.8, 0.84610)	**(1335.8, 0.90048)**	28.4	(22229.9, 0.95234)	(7797.8, 0.94846)	-54.9	(4885.1, 0.93406)	**(4137.6, 0.93877)**	-25.5
	108	(1335.8, 0.84508)	**(1335.8, 0.90251)**	28.4	(44389.8, 0.95234)	(11617.2, 0.94738)	-54.4	(6556.8, 0.93440)	(4108.4, 0.93349)	-15.6
R_{546}	24	(1122.0, 0.87705)	**(1122.0, 0.87737)**	36.4	(39858.0, 0.93076)	(20513.0, 0.92717)	54.8	(4453.0, 0.88868)	**(2628.0, 0.89193)**	16.5
	45	(629.0, 0.87248)	**(627.0, 0.87358)**	57.7	(39858.0, 0.94196)	(20805.0, 0.93659)	-49.8	(2920.0, 0.89700)	**(2263.0, 0.90978)**	1.44
	66	(627.0, 0.87567)	**(627.0, 0.88199)**	46.5	(19929.0, 0.95710)	(8541.0, 0.95214)	-51.3	(2847.0, 0.90954)	**(1533.0, 0.91425)**	5.8
	87	(627.0, 0.87032)	**(627.0, 0.88323)**	49.3	(19929.0, 0.95710)	(6935.0, 0.95350)	-56.0	(2409.0, 0.91396)	**(1347.0, 0.91838)**	2.91
	108	(627.0, 0.86827)	**(627.0, 0.89088)**	39.7	(39858.0, 0.95710)	(10804.0, 0.95271)	-47.2	(2628.0, 0.91306)	**(1274.0, 0.91814)**	7.3
Di_{529}	24	(1881.0, 0.87795)	**(1881.0, 0.87919)**	51.2	(38617.0, 0.93284)	(19418.0, 0.92927)	-63.2	(29492.0, 0.93114)	(19418.0, 0.92927)	-63.2
	45	(1571.1, 0.86558)	**(1571.1, 0.87024)**	66.7	(38617.0, 0.94371)	(20367.0, 0.93856)	-52.1	(31828.0, 0.94131)	(20367.0, 0.93856)	-52.1
	66	(1505.1, 0.86760)	(1505.2, 0.87470)	65.3	(19418.0, 0.95841)	(8833.0, 0.95399)	-54.6	(16133.0, 0.95778)	(8833.0, 0.95399)	-54.6
	87	(1485.1, 0.86033)	(1485.2, 0.87114)	69.7	(19418.0, 0.95841)	(7081.0, 0.95513)	-59.6	(13067.0, 0.95719)	7081.0, 0.95513)	-59.6
	108	(1485.1, 0.86340)	(1485.2, 0.87542)	66.1	(38617.0, 0.95841)	(10293.0, 0.95412)	-57.8	(22630.0, 0.95688)	(10293.0, 0.95412)	-57.8

same as BSAMod. A set of similar experiments was carried out considering a varying number of processors, for the DAGs G_{1034}, R_{546} and Di_{529}. The main conclusion is that the improvements on the makespan provided by MRCD were significant at a cost of a small deterioration of the reliability for any number of processors, as can be seen in Table 5.

Table 4. Comparison between BSAMod and MRCD for G_{702}

DAG	0.00	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00	Average
M(%)	0.00	18.50	49.00	75.00	106.00	142.00	113.00	10.00	17.00	13.00	0.00	49.4
R_T (%)	0.00	2.80	3.90	5.60	7.80	8.20	5.20	-0.80	0.30	0.20	0.00	3.01

Finally, comparing all the solutions generated by MRCD with BSAMod for the values in W for G_{152}, G_{702}, Di_{81}, Di_{361} on 24 processors, it could be observed that all MRCD solutions were dominant or incomparable. In the case of R_{80} and R_{364}, in only one case MRCD was dominated by BSAMod. In summary, one can conclude the benefits of applying the implemented list scheduling algorithm together with the cost function $f(v, p_v)$ in MRCD. Actually, it was observed that,

Table 5. Comparison between BSAMod and MRCD for a varying number of processors

	G_{1034}			R_{546}			Di_{529}		
m	24	87	213	24	87	213	24	87	213
Avg. $\mathcal{M}(\%)$	47.2	41.1	41.1	-3.34	5.55	8.25	105	71.5	68.3
Avg. R_T (%)	0.61	1.41	1.28	-0.13	2.04	0.52	1.23	3.08	2.79

since the normalization procedure of BSAmod only divides by the maximum value of the given objective, it leaded to poor choices.

5 Concluding Remarks

This work proposes a weighted cost function and a classification of the solutions provided by MRCD. An experimental analysis shows that MRCD can find efficient solutions when compared with the other heuristics under evaluation. The comparison with HEFT shows the importance of using a bi-objective function that considers both minimization of the makespan and maximization of the reliability. From the evaluation of RCDMod and MRCD, one can conclude the importance of using an integrated bi-objective function, while the importance of the weights and the cost function provided by MRCD are shown when comparing with RHEFT. Finally, when considering BSA, which is also a weighted bi-objective scheduling heuristics, MRCD provides outstanding improvements on the makespan while keeping similar reliability results. As future work, MRCD will be used in conjunction with a fault tolerance approach that is primary-backup based, with the objective to recover real MPI applications from faults and execute them efficiently on distributed heterogeneous systems.

References

1. Qin, X., Jiang, H.: A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. Parallel Computing 32(5), 331–356 (2006)
2. Dongarra, J., Jeannot, E., Saule, E., Shi, Z.: Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In: Proc. 19th Annual ACM Symp. on Parallelism in Algorithms and Architectures (SPAA '07). ACM Press, New York (2007)
3. Topcuouglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. 13(3), 260–274 (2002)
4. Hakem, M., Butelle, F.: Reliability and scheduling on systems subject to failures. In: Proc. of the Int. Conf.e on Parallel Processing (ICPP), p. 38 (2007)
5. Gal, T., Hanne, T., Stewart, T. (eds.): Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory, and Applications. Kluwer Academic, Dordrecht (1999)
6. Nascimento, A.P., Sena, A.C., Boeres, C., Rebello, V.E.F.: Distributed and dynamic self-scheduling of parallel MPI grid applications. Concurrency and Computation: Practice and Experience 19(14), 1955–1974 (2007)
7. Girault, A., Saule, 1., Trystram, D.: Reliability versus performance for critical applications. J. Parallel and Distrib. Computing 69(3), 326–336 (2009)

Two-Dimensional Matrix Partitioning for Parallel Computing on Heterogeneous Processors Based on Their Functional Performance Models

Alexey Lastovetsky and Ravi Reddy

School of Computer Science and Informatics, University College Dublin,
Belfield Dublin 4, Ireland
{Alexey.Lastovetsky,Manumachu.Reddy}@ucd.ie

Abstract. The functional performance model (FPM) of heterogeneous processors has proven to be more realistic than the traditional models because it integrates many important features of heterogeneous processors such as the processor heterogeneity, the heterogeneity of memory structure, and the effects of paging. Optimal 1D matrix partitioning algorithms employing FPMs of heterogeneous processors are already being used in solving complicated linear algebra kernel such as dense factorizations. However, 2D matrix partitioning algorithms for parallel computing on heterogeneous processors based on their FPMs are unavailable. In this paper, we address this deficiency by presenting a novel iterative algorithm for partitioning a dense matrix over a 2D grid of heterogeneous processors and employing their 2D FPMs. Experiments with a parallel matrix multiplication application on a local heterogeneous computational cluster demonstrate the efficiency of this algorithm.

Keywords: data partitioning algorithms, functional performance models, heterogeneous processors, parallel matrix multiplication.

1 Introduction

Traditional data partitioning algorithms for parallel computing on heterogeneous processors [1-3] are based on a performance model, which represents the speed of a processor by a constant positive number and computations are distributed amongst the processors such that their volume is proportional to this speed of the processor. The number characterizing the performance of the processor is typically its relative speed demonstrated during the execution of a serial benchmark code solving locally the core computational task of some given size.

The traditional constant performance models (CPMs) proved to be accurate enough for heterogeneous distributed memory systems if partitioning of the problem results in a set of computational tasks that fit into the main memory of the assigned processor. But these models become less accurate in the presence of paging. The functional performance model (FPM) of heterogeneous processors proposed and analysed in [3] has proven to be more realistic than the CPMs because it integrates many important

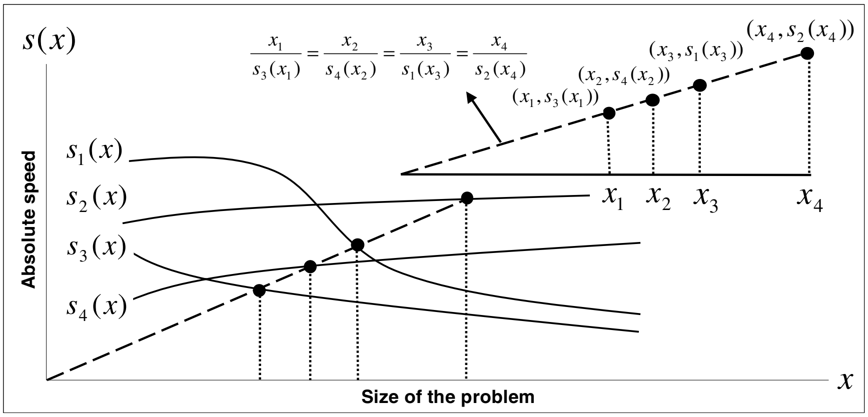


Fig. 1. Optimal data distribution showing the geometric proportionality of the number of chunks to the speed of the processor

features of heterogeneous processors such as the processor heterogeneity, the heterogeneity of memory structure, and the effects of paging. The algorithms employing it therefore distribute the computations across the heterogeneous processors more accurately than the algorithms employing the CPMs. Under this model, the speed of each processor is represented by a continuous function of the size of the problem. This model is application centric because, generally speaking, different applications will characterize the speed of the processor by different functions.

The problem of distributing independent chunks of computations over a one-dimensional arrangement of heterogeneous processors using this FPM has been studied in [4]. It can be formulated as follows: Given n independent chunks of computations, each of equal size (i.e., each requiring the same amount of work), how can we assign these chunks to p ($p<n$) physical processors P_1, P_2, ..., P_p with their respective full FPMs represented by speed functions $s_1(x)$, $s_2(x)$, ..., $s_p(x)$ so that the workload is best balanced? An algorithm solving it with a complexity of $O(p \times \log_2 n)$ is also presented. This and other similar algorithms, which relax the restriction of bounded heterogeneity of the processors [5] and which are not sensitive to the shape of speed functions [6], are based on the observation that the optimal data distribution points $(x_1, s_1(x_1))$, $(x_2, s_2(x_2))$, …, $(x_p, s_p(x_p))$ lie on a straight line passing through the origin of the coordinate system and are the intersecting points of this line with the graphs of the speed functions of the processors. This is shown in Figure 1. These algorithms are used as building blocks in algorithms solving more complicated linear algebra kernel such as the dense factorizations [7].

However, 2D matrix partitioning algorithms for parallel computing on heterogeneous processors and employing their FPMs are unavailable. We address this deficiency in this paper by presenting a novel iterative algorithm of optimal 2D data partitioning for parallel computing on a 2D grid of heterogeneous processors and employing their 2D FPMs. The algorithm assumes the 2D FPMs are given. Using experimental results with parallel matrix multiplication applications on a local heterogeneous computational cluster, we demonstrate the efficiency of this algorithm.

The rest of the paper is organized as follows. In Section 2, we present the contribution of this paper, which is the data partitioning algorithm. This is followed by experimental results on a local heterogeneous computing cluster in Section 3. For the experiments, we use parallel matrix multiplication applications demonstrating the efficiency of the algorithm.

2 Data Partitioning Algorithm

The data partitioning problem that we are trying to solve can be formulated as follows:

- Given
  - The problem size represented by a set of two parameters, (m,n), i.e., $m \times n$ independent chunks of computations each of equal size (i.e., each requiring the same amount of work). The problem size characterizes the amount and layout of data in two dimensions, for example, a dense matrix of size $m \times n$;
  - (p,q), the dimensions representing the 2D processor grid of size $p \times q$, P_{ij}, $i \in [1,p], j \in [1,q]$
  - The FPMs of the processors, $S=\{s_{ij}(x,y)$, $i \in [1,p], j \in [1,q]$. The execution time t to execute a problem size (x,y) on a processor i can be calculated using the formula $(x \times y)/s_i(x,y)$;
  - ε, the termination criterion, which represents the required relative accuracy of the solution.
- Assign each processor, P_{ij}, a block of r_{ij} rows and c_{ij} columns satisfying the conditions, $\sum_{i=1}^{p} r_{ij} = m, \forall j \in [1,q]$ and $\sum_{j=1}^{q} c_{ij} = n, \forall i \in [1,p]$, meaning that it is responsible for computing $r_{ij} \times c_{ij}$ computational chunks, such that
  - The rectangular partitions, $r_{ij} \times c_{ij}$ form a two-dimensional $p \times q$ arrangement, and
  - The maximum relative difference (MRD) between execution times on the processors is less than or equal to ε, i.e., MRD≤ε.

***Data Partitioning Algorithm (DPA-FPM-2D)*:** The inputs to the algorithm are

- The problem size represented by a set of two parameters, (m,n);
- (p,q), the dimensions representing the 2D processor grid of size $p \times q$, P_{ij}, $(i \in [1,p], j \in [1,q])$;
- The FPMs of the processors, $S=\{s_{ij}(x,y)\}$;
- The termination criterion, ε.

The outputs r and c are integer arrays of size $p \times q$. The (i,j)-th element of r is the number of rows allocated to processor P_{ij} and the (i,j)-th element of c is the number of columns allocated to processor P_{ij}. The algorithm can be summarized as follows:

- **Initialization:**
  - The execution times to execute the problem size, (m/p, n/q), on all the processors, P_{ij} ($i \in [1,p], j \in [1,q]$), are calculated using the formula $t_{ij}^1 = \left(\left(m/p\right)\times\left(n/q\right)\right)/s_{ij}^1(m/p, n/q)$. If MRD≤ε, the algorithm terminates. The optimal distribution is $r_{ij}=m/p$, $c_{ij}=n/q$;
  - Otherwise, the algorithm proceeds to the next step, for which the inputs are the single-number speeds, $s_{ij}^1\left(r_{ij}^1, c_{ij}^1\right)$ where $r_{ij}^1 = m/p$, $c_{ij}^1 = n/q$.
- **Iteration:** At step k+1,
  - The procedure, *HCOL*(m, n, p, q, s) illustrated in Figure 2, is invoked to determine the column-based data distribution, which is optimal for the single-number speeds, $s = s_{ij}^k\left(r_{ij}^k, c_{ij}^k\right)$. The resulting data distribution is $\left(r_{ij}^{k+1}, c_{ij}^{k+1}\right)$;
  - The execution times, t_{ij}^{k+1}, are then calculated using the formula $t_{ij}^{k+1} = \left(r_{ij}^{k+1}\times c_{ij}^{k+1}\right)/s_{ij}^{k+1}(r_{ij}^{k+1}, c_{ij}^{k+1})$. If MRD≤ε, the algorithm terminates. The optimal distribution is $(r_{ij}^{k+1}, c_{ij}^{k+1})$. If MRD>ε, the algorithm proceeds to the next step;
  - The MRD in each processor column is checked. For each of the processor columns where MRD>ε, the algorithm executes the procedure, *HSPF*(m, c, p, S), which returns the optimal distribution of m independent chunks over p heterogeneous processors P_i of respective speed functions S={$s_{ij}(x,c)$} where c is a constant. Here c is the size of the column block, which is the same for all the processors in a processor column. For the sake of simplicity, let us assume the existence of one such column x. The inputs to the procedure are then m, c_{1x}^{k+1}, p, $S=\{s_i(x,c_{1x}^{k+1})\}_{i=1}^p$. The resulting data distribution is $\left(r_{ix}^{k+1}\right), \forall i \in [1,p]$. The algorithm then proceeds to the next iteration, for which the inputs from this processor column are the speeds, $s_{ix}^{k+1}(r_{ix}^{k+1}, c_{1x}^{k+1}), \forall i \in [1,p]$. For the processor columns for which the MRD≤ε, the inputs are the speeds, $s_{ij}^{k+1}(r_{ij}^{k+1}, c_{ij}^{k+1})$.

***HCOL*:** This procedure invokes the data partitioning algorithm [1,3], which determines the optimal 2D column-based partitioning of a dense matrix of size m×n on a 2D heterogeneous processor grid of size p×q. The matrix is partitioned into uneven rectangles so that they are arranged into a 2D grid of size p×q and the area of a rectangle is proportional to the speed of the processor owning it. The inputs to the algorithm are

- The dense matrix of size m×n;
- (p,q), the dimensions representing the 2D processor grid of size p×q, P_{ij}, $i \in [1,p], j \in [1,q]$;
- The single number speeds of the processors, s_{ij}.
- The output is the heights and the widths of the rectangles, (r_{ij},c_{ij}).

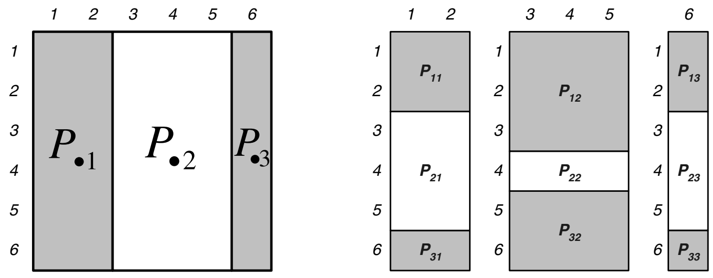


Fig. 2. Example of two-step distribution of a 6x6 square over a 3x3 processor grid. The relative speed of processors is given by {0.11, 0.25, 0.05, 0.17, 0.09, 0.08, 0.05, 0.17, 0.03}. (a) At the first step, the 6x6 square is distributed in a one-dimensional block fashion over processors columns of the 3x3 processor grid in proportion 0.33:0.51:0.16 ≈ 2:3:1. (b) At the second step, each vertical rectangle is distributed independently in a one-dimensional block fashion over processors of its column. The first rectangle is distributed in proportion 0.11:0.17:0.05 ≈ 2:3:1. The second one is distributed in proportion 0.25:0.09:0.17 ≈ 3:1:2. The third is distributed in proportion 0.05:0.08:0.03 ≈ 2:3:1.

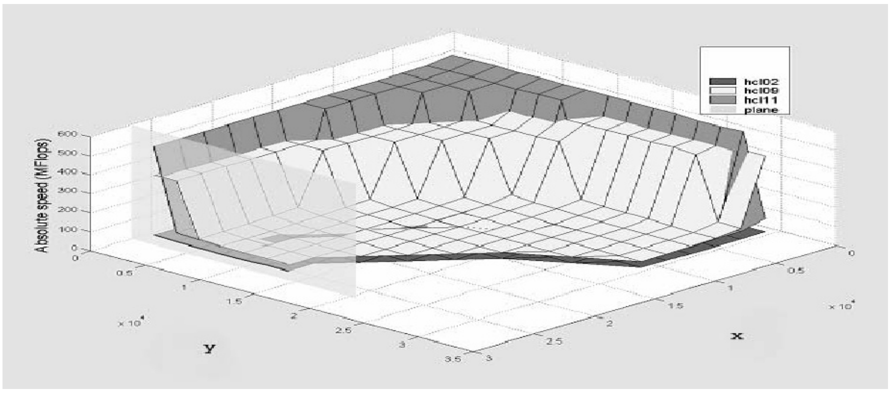
The algorithm can be summarized as follows:

- First, the area $m \times n$ is partitioned into q vertical slices, so that the area of the j-th slice is proportional to $\sum_{i=1}^{p} s_{ij}$ (see Figure 2(a)). It is supposed that blocks of the j-th slice will be assigned to processors of the j-th column in the $p \times q$ processor grid. Thus, at this step, the load *between* processor columns in the $p \times q$ processor grid is balanced, so that each processor column will store a vertical slice whose area is proportional to the total speed of its processors;
- Then, each vertical slice is partitioned independently into p horizontal slices, so that the area of the i-th horizontal slice in the j-th vertical slice is proportional to s_{ij} (see Figure 2(b)). It is supposed that blocks of the i-th horizontal slice in the j-th vertical slice will be assigned to processor P_{ij}. Thus, at this step, the load of processors *within* each processor column is balanced independently.

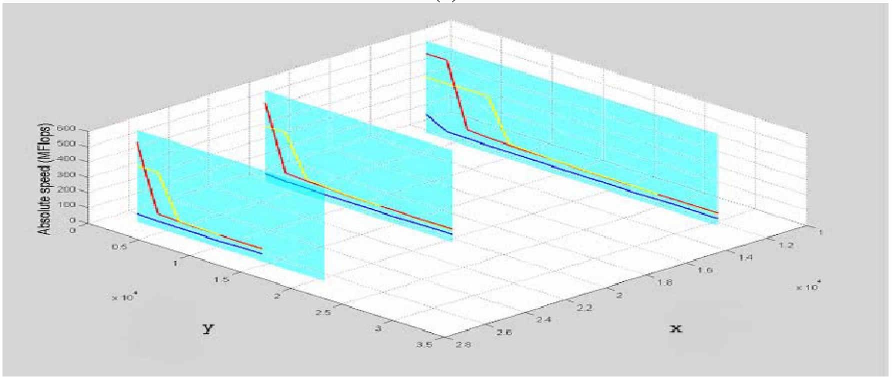
***HSPF*:** This procedure returns the optimal distribution of m independent chunks over p heterogeneous processors of $P_1, P_2, ..., P_p$ of respective speeds $S=\{s_1(x,y), s_2(x,y), ..., s_p(x,y)\}$ (HSPF stands for Heterogeneous Set Partitioning using Functional model of heterogeneous processors). It is composed of two steps:

- Surfaces, $z_i=s_i(x,y)$, representing the absolute speeds of the processors are sectioned by the plane $x=c$ (as shown on Figure 3 (a) for 3 surfaces). A set of p curves on this plane (as shown in Figure 3 (b)) will represent the absolute speeds of the processors against variable y given parameter x is fixed;
- Apply the set partitioning algorithm [4] to this set of p curves to obtain an optimal distribution.

The DPA-FPM-2D algorithm can be intuitively explained as follows. The goal of the algorithm is to determine points $(r_{ij},c_{ij},s_{ij}(r_{ij},c_{ij}))$ in the (r,c,s) space such that $(r_{ij} \times c_{ij})/s_{ij}(r_{ij},c_{ij})=C$, where C is a constant (and also the execution time). Each iteration



(a)



(b)

Fig. 3. (a) Two surfaces representing the absolute speeds of 2 processors are sectioned by the planes x=c. (b) Curves on this plane represent the absolute speeds of the processors against variable y, given parameter x is fixed.

step facilitates convergence to the optimal solution along the coordinate c and the execution of *HSPF* procedure *within the processor columns* provides the optimal solution along the coordinate r by fixing the value of the coordinate c. During the execution of the procedure *HSPF*, the coordinate 'c' of the (r,c,s) space is kept constant and 1D FPMs of the processors, (r_{ix},s_{ix}(r_{ix},c_{1x})), are built against parameter r in the (r,s) plane, taking one processor column x for instance. These are a set of p curves representing the absolute speeds of the p processors against parameter r given parameter c is fixed. It can be visualized as the plane y=c intersecting the speed surfaces in the (r,c,s) space to give p curves in the (r,s) plane. The data partitioning algorithm [4] is then applied to this set of p curves to obtain optimal data distribution for these 1D FPMs so that $r_{ix}/s_{ix}(r_{ix},c_{1x})=C_1$, $i \in [1, p]$,where C_1 is a constant. Since c_{1x} is the same for all the processors in the column x, $(r_{ix}\times c_{1x})/s_{ix}(r_{ix},c_{1x})=C_2$ $\forall i \in [1, p]$, where C_2 is a constant too.

The speeds from the new data distribution are employed again to determine the new value of the coordinate c, which would be closer to the optimal solution, and the iteration procedure is repeated. Thus in this manner, the algorithm converges to the optimal solution.

3 Experimental Results

We use a small heterogeneous local network of 16 different Linux processors (hcl01-hcl16) for the experiments. The specifications of the network are available at the URL http://hcl.ucd.ie/Hardware/Cluster+Specifications. The network is based on 2 Gbit Ethernet with a switch enabling parallel communications between the computers. The software used is MPICH-1.2.5.

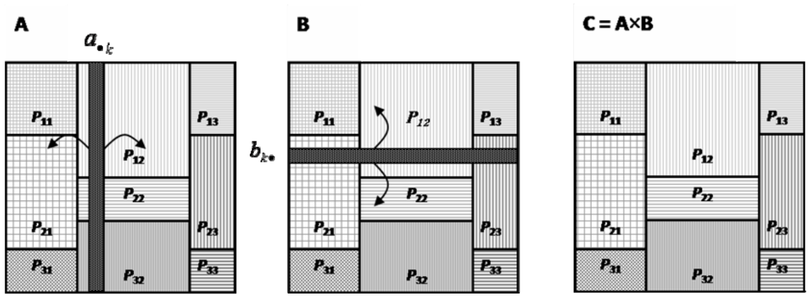


Fig. 4. One step of the algorithm of parallel matrix multiplication employing a 2D heterogeneous processor grid of size 3x3. Matrices A, B, and C are partitioned such that the area of the rectangle is proportional to the speed of the processor owning it. First, each $b{\times}b$ block of the pivot column $a_{\bullet k}$ of matrix A (shown with curly arrows) is broadcast horizontally, and each $b{\times}b$ block of the pivot row $b_{k\bullet}$ of matrix B (shown with curly arrows) is broadcast vertically. Then, each $b{\times}b$ block c_{ij} of matrix C is updated, $c_{ij}=c_{ij}+a_{ik}{\times}b_{kj}$.

Figure 4 shows the parallel matrix multiplication application used for the experiments. It implements the matrix operation $C=A{\times}B$, multiplying matrix A and matrix B, where A, B, and C are dense matrices of size $m{\times}k$, $k{\times}n$, and $m{\times}n$ matrix elements respectively on a 2D heterogeneous processor grid of size $p{\times}q$. We use dense square matrices and a 2D heterogeneous processor grid of size 3x3 for illustration purposes. Each matrix element is a square block of size $b{\times}b$ (value of b used in the experiments is 64). The heterogeneous parallel algorithm used to compute this matrix product is a modification of the ScaLAPACK outer-product algorithm [8]. We assume that only one process is configured to execute on a processor. The data partitioning problem is that the matrices A, B, and C must be divided into rectangles such that there is one-to-one mapping between the rectangles and the processors, and the area of each rectangle is proportional to the speed of the processor owning it.

For this application, the core computational kernel performs a matrix update of a matrix C_b of size $m_b{\times}n_b$ using A_b of size $m_b{\times}1$ and B_b of size $1{\times}n_b$ as shown in Figure 5. The size of the problem is represented by two parameters, m_b and n_b. We use a combined computation unit, which is made up of one addition and one multiplication, to express the volume of computation. Therefore, the total number of computation units (namely, multiplications of two b×b matrices) performed during the execution of the benchmark code will be approximately equal to m_b×n_b. Therefore, the absolute speed of the processor exposed by the application when solving the problem of size (m_b,n_b) can be calculated as m_b×n_b divided by the execution time of the matrix update. This gives us a function, f: $\mathbf{N}^2 \rightarrow \mathbf{R}_+$, mapping problem sizes to speeds of the processor. The FPM of the processor is obtained by continuous extension of function f: $\mathbf{N}^2 \rightarrow \mathbf{R}_+$ to function g: $\mathbf{R}_+^2 \rightarrow \mathbf{R}_+$ (f(n,m)=g(n,m) for any (n,m) from $\mathbf{N}^2$).

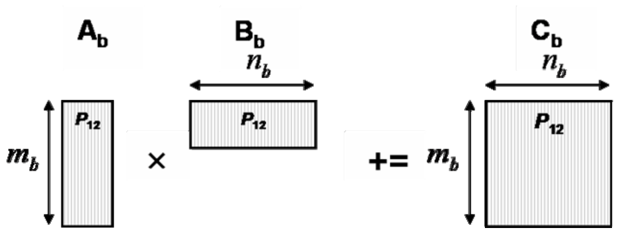


Fig. 5. The computational kernel (shown here for processor P_{12} for example) performs a matrix update of a dense matrix C_b of size m_b×n_b using A_b of size m_b×1 and B_b of size 1×n_b. The matrix elements represent b×b matrix blocks.

The heterogeneity of the network due to the heterogeneity of the processors is calculated as the ratio of the absolute speed of the fastest processor to the absolute speed of the slowest processor. For example, consider the benchmark code of a local DGEMM update of two matrices 2560×64 and 64×2560, the absolute speeds of the processors hcl01-hcl16 in million flop/s performing this update are {130, 258, 188, 188, 188, 214, 125, 127, 157, 232, 147, 137, 157, 197, 194, 201}. As one can see, hcl02 is the fastest processor and hcl07 is the slowest processor. The heterogeneity is therefore 2.

Figure 6 shows the execution times of the sequential application and three parallel applications solving the same matrix multiplication problem. The execution of the parallel applications consists of two parts. First, all the processors execute a data partitioning algorithm to partition the matrices and then they perform the parallel matrix multiplication itself.

The first parallel application employs a data partitioning algorithm (DPA-CPM-2D) that uses the CPMs of the processors [1,3]. The constant single number speeds of the processors are calculated from the execution of a local DGEMM update of two matrices of sizes, (m/p)×b and b×(n/q) respectively where (m,n) is the problem size, p=4, q=4, and b=64. The second parallel application employs a data partitioning algorithm DPA-FPM-1D that optimally partitions the matrix over a 1D arrangement of processors based on their FPMs. The inputs to DPA-FPM-1D are the problem size (m, n), the 1D arrangement of 16 processors and their FPMs. The third parallel application employs DPA-FPM-2D. The parameters to DPA-FPM-2D are the problem size (m, n), p=4, q=4, ε=0.05, and the full FPMs of the processors.

The sequential application is executed on the fastest processor (hcl02). For problem sizes ((m,n), m>10240 & n>10240), the sequential application fails due to the problem

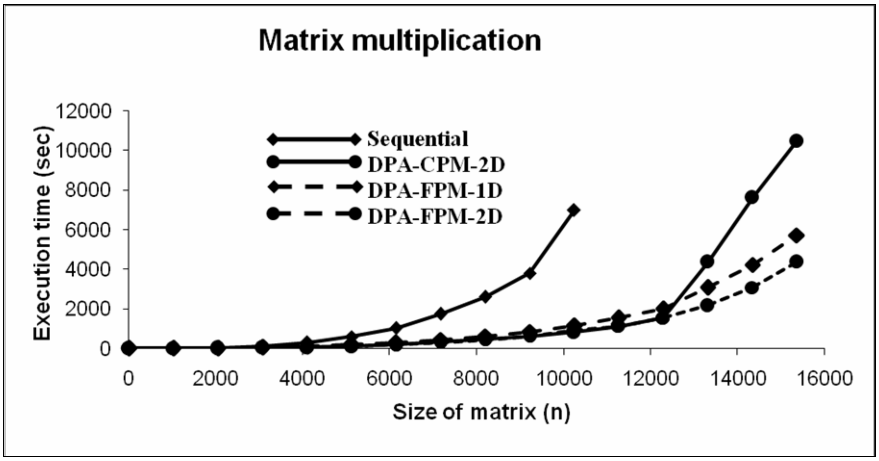


Fig. 6. Execution times of the parallel application employing DPA-FPM-2D, a parallel application employing 1D grid of heterogeneous processors and their associated FPMs, a parallel application employing a data partitioning algorithm using constant single-number speeds and a sequential application solving the same matrix multiplication problem.

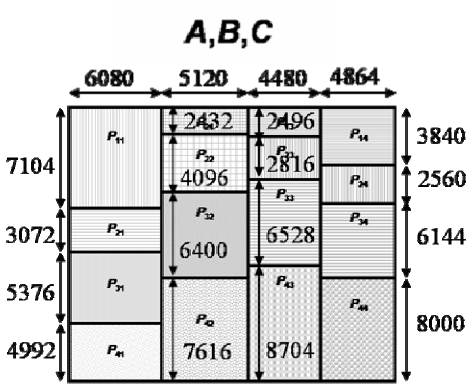


Fig. 7. Optimal data distribution using DPA-FPM-2D whose parameters are m=n=20544, p=4, q=4

size exceeding the memory limit of the processor. One or more processors start paging around the problem sizes ((m,n), 1≤m≤12288 & 1≤n≤12288). The parallel application employing the DPA-CPM-2D algorithm fails for problem sizes ((m,n), m>15360 & n>15360).

Figure 7 shows the optimal data distribution of the dense matrices A, B, and C determined by DPA-FPM-2D for the problem size (m,n)=(20544,20544). One can see that the parallel application employing the DPA-FPM-2D algorithm outperforms the other applications. The total number of iteration steps of DPA-FPM-2D observed are 1 for the problem sizes, ((m,n), 1≤m≤12288 & 1≤n≤12288) and 2 for the problem sizes in the region of paging, ((m,n), m>12288 & n>12288). Therefore, we can conclude the DPA-FPM-2D algorithm converges very fast. Its execution time is also found to be several orders of magnitude less than the execution time of the parallel matrix multiplication.

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/IN.1/I2054.

References

[1] Kalinov, A., Lastovetsky, A.: Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers. Journal of Parallel and Distributed Computing 61(4), 520–535 (2001)

[2] Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Matrix Multiplication on Heterogeneous Platforms. IEEE Transactions on Parallel and Distributed Systems 12(10), 1033–1051 (2001)

[3] Lastovetsky, A., Reddy, R.: On Performance Analysis of Heterogeneous Parallel Algorithms. Parallel Computing 30(11), 1195–1216 (2004)

[4] Lastovetsky, A., Reddy, R.: Data Partitioning with a Functional Performance Model of Heterogeneous Processors. International Journal of High Performance Computing Applications 21(1), 76–90 (2007)

[5] Lastovetsky, A., Reddy, R.: Data Partitioning for Multiprocessors with Memory Heterogeneity and Memory Constraints. Scientific Programming 13(2), 93–112 (2005)

[6] Lastovetsky, A., Reddy, R.: Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers. In: 17th International Parallel and Distributed Processing Symposium. IEEE Computer Society Press, Los Alamitos (2004)

[7] Lastovetsky, A., Reddy, R.: Data distribution for dense factorization on computers with memory heterogeneity. Parallel Computing 33(12), 757–779 (2007)

[8] Petitet, A., Dongarra, J.: Algorithmic Redistribution Methods for Block-Cyclic Decompositions. IEEE Transactions on Parallel and Distributed Systems 10(12), 1201–1216 (1999)

Accelerating S3D: A GPGPU Case Study

Kyle Spafford[1], Jeremy Meredith[1], Jeffrey Vetter[1],
Jacqueline Chen[2], Ray Grout[2], and Ramanan Sankaran[1]

[1] Oak Ridge National Laboratory
{spaffordkl,jsmeredith,vetter,sankaranr}@ornl.gov
[2] Sandia National Laboratories
{jhchen,rwgrout}@sandia.gov

Abstract. The graphics processor (GPU) has evolved into an appealing choice for high performance computing due to its superior memory bandwidth, raw processing power, and flexible programmability. As such, GPUs represent an excellent platform for accelerating scientific applications. This paper explores a methodology for identifying applications which present significant potential for acceleration. In particular, this work focuses on experiences from accelerating S3D, a high-fidelity turbulent reacting flow solver. The acceleration process is examined from a holistic viewpoint, and includes details that arise from different phases of the conversion. This paper also addresses the issue of floating point accuracy and precision on the GPU, a topic of immense importance to scientific computing. Several performance experiments are conducted, and results are presented from the NVIDIA Tesla C1060 GPU. We generalize from our experiences to provide a roadmap for deploying existing scientific applications on heterogeneous GPU platforms.

1 Introduction

Strong market forces from the gaming industry and increased demand for high definition, real-time 3D graphics have been the driving forces behind the GPU's incredible transformation. Over the past several years, increases in the memory bandwidth and the speed of floating point computation of GPUs have steadily outpaced those of CPUs. In a relatively short period of time, the GPU has evolved from an arcane, highly-specialized hardware component into a remarkably flexible and powerful parallel coprocessor.

1.1 GPU Hardware

Originally, GPUs were designed to perform a limited collection of operations on a large volume of independent geometric data. These operations fell into to only two main categories (vertex and fragment) and were highly parallel and computationally intense, resulting in a highly specialized design with multiple cores and small caches. As graphical tasks became more diverse, the demand for flexibility began to influence GPU designs. GPUs transitioned from a fixed function design, to one which allowed limited programmability of its two specialized pipelines, and eventually to an approach where all its cores were of a unified, more flexible type, supporting much greater control from the programmer.

1.2 CUDA

The striking performance numbers of modern GPUs have resulted in a surge of interest in general-purpose computation on graphics processing units (GPGPU). GPGPU represents an inexpensive and power-efficient alternative to more traditional HPC platforms. In the past, there has been a substantial learning curve associated with GPGPU, and expert knowledge was required to attain impressive performance. This involved extensive modification of traditional approaches in order to effectively scale to the large number of cores per GPU. However, as the flexibility of the GPU has increased, there has been a welcomed decrease in the associated learning curve of the porting process. In this study, we utilize NVIDIA's Compute Unified Device Architecture (CUDA), a parallel programming model and software environment. CUDA exposes the power of the GPU to the programmer through a set of high level language extensions, allowing for existing scientific codes to be more easily transformed into GPU compatible applications.

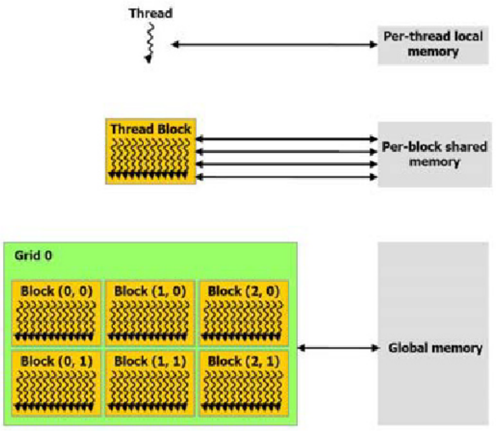


Fig. 1. CUDA Programming Model – Image from NVIDIA CUDA Programming Guide[1]

Programming Model. While a full introduction to CUDA is beyond the scope of this paper, this section mentions the basic concepts required to understand the scope of the parallelism involved. CUDA views the GPU as a highly parallel coprocessor. Functions called kernels, are composed of a large number of threads, which are organized into blocks. A group of blocks is known as a grid, see Figure 1. Blocks contain a fast shared memory that is only available to threads which belong to the block, while grids have access to the global GPU memory. Typical kernel launches involve one grid, which is composed of hundreds or thousands of individual threads, a much higher degree of parallelism than normally occurs with traditional parallel approaches on the CPU. This high degree of parallelism and unique memory architecture have drastic consequences for performance, which will be explored in a later section.

1.3 Domain and Algorithm Description

S3D is a massively parallel direct numerical solver (DNS) for the full compressible Navier-Stokes, total energy, species and mass continuity equations coupled with detailed chemistry[2, 3]. It is based on a high-order accurate, non-dissipative numerical scheme solved on a three-dimensional structured Cartesian mesh. S3D's performance has been studied and optimized including I/O[4] and control flow[5]. Still, further improvements allow for increased grid size, more simulation timesteps, and more species equations. These are critical to the scientific goals of turbulent combustion simulations in that they help achieve higher Reynolds numbers, better statistics through larger ensembles, more complete temporal development of a turbulent flame, and the simulation of fuels with greater chemical complexity.

Here we assess S3D code performance and parallel scaling through simulation of a small amplitude pressure wave propagating through the domain for a short period of time. The test is conducted with detailed ethylene-air (C_2H_4) chemistry consisting of twenty-two chemical species and mixture-averaged molecular transport model. Due to the detailed chemical model, the code solves for twenty-two species equations in addition to the five fluid dynamic variables.

2 Related Work

Recent work by a number of researchers has investigated GPGPU with impressive results in a variety of domains. Owens et. al. provide an excellent history of the GPU [6], chronicling its transformation in great detail. It is not uncommon to find researchers who achieve at least an order of magnitude improvement over reference implementations. GPUs have been used to accelerate a variety of application kernels, including more traditional operations like dense[7, 8, 9] and sparse[10] linear algebra as well as scatter-gather techniques[11]. The GPU has been successfully applied to a wide variety of fields including computational biophysics[12], molecular dynamics[13], and medical imaging[14, 15]. Our work takes a slightly higher level approach. While we do present performance measurements from an accelerated version of S3D, we examine the acceleration process as a whole, and endeavor to answer why certain applications perform so well on GPUs, while others fail to achieve significant performance improvements.

3 Identifying Candidates for Acceleration

3.1 Profiling

The first step in identifying a scientific application for acceleration is to identify the performance bottlenecks. The best case scenario involves a small number of computationally intense functions which comprise most of the runtime.

This is a fairly basic requirement and is a direct consequence of Amdahl's law. The CPU based profiling tool Tau identified S3D's getrates kernel as a major bottleneck[16]. This kernel involves calculating the rates of chemical reactions occurring in the simulation at each point in space. This computation comprises about half of the total runtime with the current chemistry model. As the chemical model becomes more complex, we anticipate that the getrates kernel will begin to comprise a stronger majority of total runtime. As the kernel's total percentage of runtime increases, the greater the potential for application speedup. Therefore, when choosing kernels to accelerate, the first to be examined should be the most time consuming.

3.2 Parallelism and Data Dependency

One of the main advantages of the GPU is the high number of processors, so it follows that kernels must exhibit a high degree of parallelism to be successful on a heterogenous GPU platform. While this can correspond to task-based parallelism, GPUs have primarily been used for data-parallel operations. This makes it difficult for GPUs to handle unstructured kernels, or those with intricate patterns of data dependency. Indeed, in situations with irregular control flow, individual threads can become serialized, which results in performance loss. Since the memory architecture of a GPU is dramatically different than most CPUs, memory access times can differ by several orders of magnitude based on access pattern and type of memory. For example, on the Tesla, an access to shared block memory is much faster than an access to global memory. Therefore, kernels must often be chosen based on memory access pattern, or restructured such that memory access is more uniform in nature. In S3D, the getrates kernel operates on a regular three dimensional mesh, so access patterns are fairly uniform, an easy case for the GPU.

The following psuedocode outlines the general structure of the sequential getrates kernel. The outer three loops can be computed in parallel, since points in the mesh are independent.

```
for x = 1 to length
   for y = 1 to length
      for z = 1 to length
         for n = 1 to nspecies
            grid[x][y][z][n] = F(grid[x][y][z][1:nspecies])
```

where length refers to the length of an edge of the cube, nspecies refers to the number of chemical species involved, and function F is an abstraction of the more complex chemical computations.

4 Kernel Acceleration

Once a suitable portion of the application has been identified, the acceleration process can begin. Parallel programming is inherently more difficult than sequential programming, and developing high performance code for GPUs also incorporates complexity from architectural features. This "memory aware" programming environment grants the programmer control over low level memory movement, but demands meticulous data orchestration to maximize performance.

For S3D, the mapping between the getrates kernel and CUDA concepts is fairly simple. Since getrates operates on a regular, three-dimensional mesh, each point in the mesh is handled by a single thread. A block is composed of a local region of the mesh. Block size was chosen to be 256, based on the available number of registers per GPU core, in order to maximize occupancy.

During the development of the accelerated version of the getrates kernel, the memory access pattern was the most important factor for performance. When threads read or write memory in a highly parallel fashion, CUDA coalesces the memory access into a single operation, which has a dramatic and beneficial effect on performance. The optimized versions of the getrates kernel also use batched memory transfers and exploit block shared memory. This attention to detail pays off–accelerated versions of the getrates kernel exhibit promising speedups over the serial CPU version: up to 14.6x for the single precision version, and 9.3x for the double precision version for a single iteration of the kernel, see Figure 2. The serial CPU version was measured on an Intel Harpertown running at 2.5Ghz with 8GB of RAM.

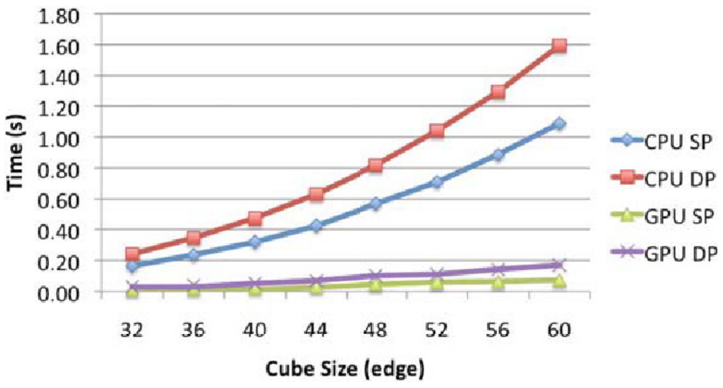


Fig. 2. Accelerated Kernel Results

5 Accuracy

While the evolution of the GPU has been remarkable, architectural remnants of its original, specialized function remain. Perhaps the most relevant of these to the scientific community is the bias towards single precision floating point computations. Single precision arithmetic was sufficient for the GPU's original tasks (rasterization, etc.). GPU benchmarking traditionally involved only these single precision computations, and performance demands have clearly shaped the GPU's allocation of hardware resources. Many GPUs are incapable of double precision, and those that are typically pay a high performance cost. This cost generally arises from the differing number of floating point units, and it is almost always more than the performance difference between single and double precision on a traditional CPU. In S3D, the cost can clearly be seen in the performance difference in the single versus double precision versions of the getrates kernel.

From a performance standpoint, single precision computations are favorable compared to double precision, but the computations in scientific applications can be extremely sensitive to accuracy. Moreover, some double precision operations are not always equivalent on the CPU and GPU. GPUs may sacrifice fully IEEE compliant floating point operations for greater performance. For example, scientific applications frequently make extensive use of transcendental functions (sin, cos, etc.), and the Tesla's hardware intrinsics for these functions are faster, but less accurate than their CPU counterparts.

5.1 Accuracy in S3D

In S3D, the reaction rates calculated by the getrates kernel are integrated over time as the simulation progresses, and error from inaccurate reaction rates compounds and propagates to other simulation variables. While this is the first comparison of double and single precision versions of S3D, the issue of accuracy has been previously studied, and some upper bounds for error are known. S3D has an internal monitor for the estimated error from integration, and can take smaller timesteps in an effort to improve accuracy. Figure 3 shows the estimated error from integration versus simulation time. In this graph, the CPU and GPU DP versions quickly begin to agree, while the single precision version is much more erratic. In both double precision versions, the internal mechanism for timestep control succeeds in settling on a timestep of appropriate size. The single precision version has a much weaker guarantee on accuracy, and the monitor has a

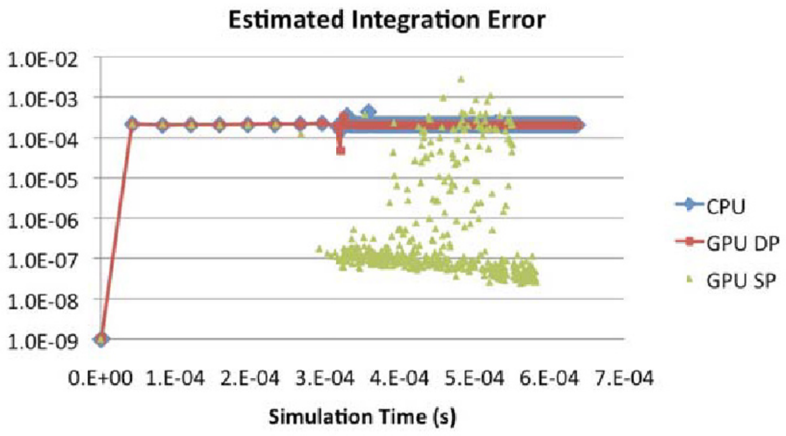


Fig. 3. Estimated Integrated Error. 1.00E-03 is the upper bound on acceptable error. The GPU DP and CPU versions completely overlap beginning roughly at time 4.00E-04.

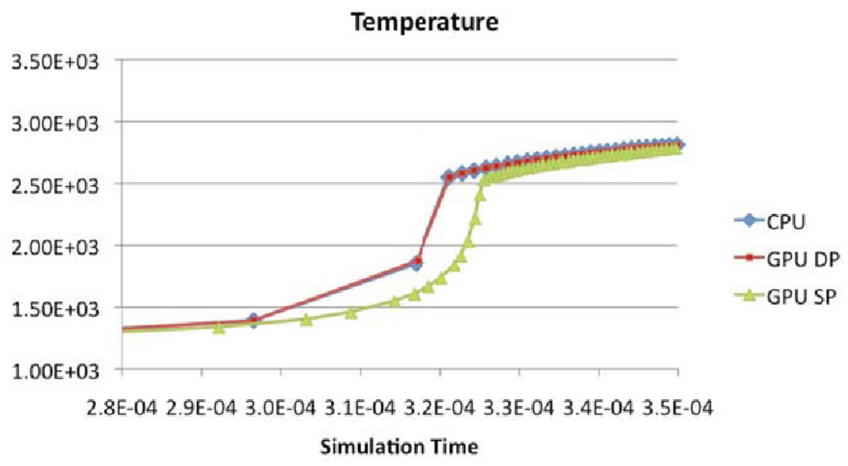


Fig. 4. Simulation temperature. Note the time gap of the increase in temperature at time roughly 3.00E-04. This corresponds to a delay in the prediction of ignition time.

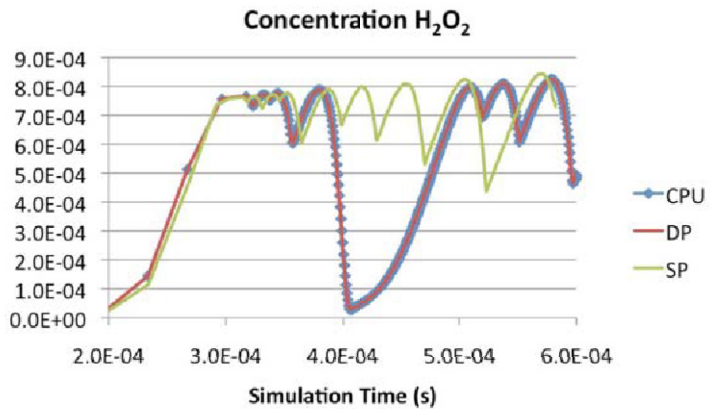


Fig. 5. Chemical Species H_2O_2. The CPU and GPU DP versions completely agree, while the GPU SP version significantly deviates, and fails to identify the dip at time 4.00E-O4.

difficult time controlling the timestep, oscillating between large timesteps with high error (sometimes beyond the acceptable bounds), and short timesteps with very low error. The increased number of timesteps required by the GPU single precision version will have consequences for performance, which will be explored in a later section.

The error from low precision can also be observed in simulation variables such as temperature (see Figure 4) or in chemical species, such as H_2O_2(see Figure 5). The current test essentially simulates a rapid ignition, and a relatively significant time gap can be seen between the rapid rise in temperature in the GPU single precision kernel versus the other versions. In the sensitive time scale

Table 1. Performance Results - Normalized cost is the average time it takes to simulate a single point in space for one nanosecond. S - Single Precision D - Double Precision.

Size	Kernel Speedup		% of	Amdahl's	Actual Speedup		Normalized Cost (ms)		
	S	D	Total	Limit	S	D	CPU	GPU DP	GPU SP
32	13.05x	8.17x	46.01%	1.82x	1.78x	1.61x	16.7	10.44	9.49
60	14.56x	9.32x	58.02%	2.38x	2.32x	2.27x	13.53	5.95	5.97

of ignition, this gap represents a serious error. In Figure 5, the error is much more pronounced, as the single precision version fails to predict the sudden decrease in H_2O_2 which occurs roughly at time 4.00E-04.

A similar trend can be observed throughout many different simulation variables in S3D. The CPU version tends to agree almost perfectly with the GPU double precision version, while the single precision version deviates substantially. Consequently, while the single precision version is much faster, it may be insufficient for sensitive simulations.

6 S3D Performance Results

In an ideal setting, the chosen kernel would strongly dominate the runtime of the application. However, in S3D, the getrates kernel comprises roughly half of the total runtime, with some variation based on problem size. Table 1 shows how speedup in the getrates kernel scales to whole-code performance improvements. Amdahl's limit is the theoretical upper bound on speedup, $s_\infty \approx \frac{1}{1-f_a}$, where f_a is the fraction of runtime that is accelerated.

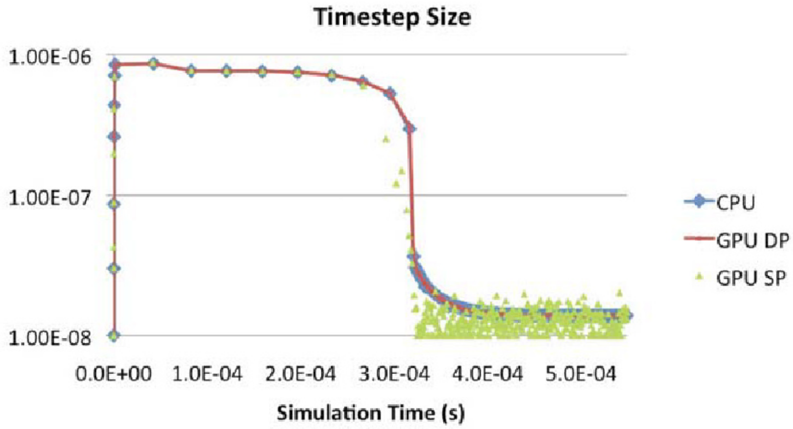


Fig. 6. Timestep Size – This graph shows the size of the timesteps taken as the rapid ignition simulation progressed. S3D reduces the timestep size when it detects integration inaccuracy. While the double precision versions take timesteps of roughly equivalent size, the single precision version quickly reduces timestep size in an attempt to preserve accuracy.

In S3D, there is a complex relationship between performance and accuracy. When inaccuracy is detected, timestep size is reduced in an attempt to decrease error, see Figure 6. Since single precision is less accurate, one can see erratic timestep sizes. This means that given the same number of timesteps, a highly accurate computation can simulate more time. In order to truly measure performance, it is important to normalize the wallclock time to account for this effect. In Table 1, normalized cost is the wallclock time it takes to simulate one nanosecond at one point in space. While the getrates kernel can be executed faster in single precision, the lack of accuracy causes the simulation to take very small timesteps. In some cases, the loss of accuracy in single precision calculations causes the total amount of simulated time to decrease, potentially eliminating any performance benefits.

7 Conclusions

Graphics processors are rapidly emerging as a viable platform for high performance scientific computing. Improvements in the programming environments and libraries for these devices are making them an appealing, cost-effective way to increase application performance. While the popularity of these devices has surged, GPUs may not be appropriate for all applications. They offer the greatest benefit to applications with well structured, data-parallel kernels. Our study has described the strengths of GPUs, and provided insights from our experience in accelerating S3D. We have also examined one of the most important aspect of GPUs for the scientific community, accuracy. The differences in accuracy between GPU and IEEE arithmetic resulted in drastic consequences for correctness in S3D. Despite this relative weakness, the heterogeneous GPU version of the kernel still manages to outperform the more traditional CPU version and produce high quality results in a real scientific application.

References

[1] NVIDIA: CUDA programming guide 2.0 downloaded (December 1, 2008), `www.nvidia.com/object/cudadevelop.html`

[2] Hawkes, E.R., Sankaran, R., Sutherland, J.C., Chen, J.H.: Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. Journal of Physics: Conference Series 16, 65–79 (2005)

[3] Sutherland, J.C.: Evaluation of mixing and reaction models for large-eddy simulation of nonpremixed combustion using direct numerical simulation. Dept. of Chemical and Fuels Engineering, PhD, University of Utah (2004)

[4] Yu, W., Vetter, J., Oral, H.: Performance characterization and optimization of parallel I/O on the Cray XT. In: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, pp. 1–11 (April 2008)

[5] Mellor-Crummey, J.: Harnessing the power of emerging petascale platforms. Journal of Physics: Conference Series 78(1), 12–48 (2007)

[6] Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., Phillips, J.: GPU computing. Proceedings of the IEEE 96(5), 879–899 (2008)

[7] Barrachina, S., Castillo, M., Igual, F., Mayo, R.: Evaluation and tuning of the level 3 CUBLAS for graphics processors. In: Proceedings of the IEEE Symposium on Parallel and Distributed Processing (IPDPS), April 2008, pp. 1–8 (2008)
[8] Fujimoto, N.: Faster matrix-vector multiplication on GeForce 8800GTX. In: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, April 2008, pp. 1–8 (2008)
[9] Cummins, G., Adams, R., Newell, T.: Scientific computation through a GPU. In: Southeastcon, pp. 244–246. IEEE, Los Alamitos (April 2008)
[10] Bolz, J., Farmer, I., Grinspun, E., Schröoder, P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In: SIGGRAPH'03: ACM SIGGRAPH 2003 Papers, pp. 917–924. ACM, New York (2003)
[11] He, B., Govindaraju, N.K., Luo, Q., Smith, B.: Efficient gather and scatter operations on graphics processors. In: SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, pp. 1–12. ACM, New York (2007)
[12] Stone, J.E., Phillips, J.C., Freddolino, P.L., Hardy, D.J., Trabuco, L.G., Schulten, K.: Accelerating molecular modeling applications with graphics processors. Journal of Computational Chemistry 28, 2618–2640 (2005)
[13] Rodrigues, C.I., Hardy, D.J., Stone, J.E., Schulten, K., Hwu, W.M.W.: GPU acceleration of cutoff pair potentials for molecular modeling applications. In: CF '08: Proceedings of the 2008 conference on Computing frontiers, pp. 273–282. ACM, New York (2008)
[14] Kruger, J., Westermann, R.: Acceleration techniques for GPU-based volume rendering. In: Visualization, VIS 2003, October 2003, pp. 287–292. IEEE, Los Alamitos (2003)
[15] Mueller, K., Xu, F.: Practical considerations for GPU-accelerated CT. In: 3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro, April 2006, pp. 1184–1187 (2006)
[16] Shende, S., Malony, A.D., Cuny, J., Beckman, P., Karmesin, S., Lindlan, K.: Portable profiling and tracing for parallel, scientific applications using C++. In: SPDT '98: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools, pp. 134–145. ACM, New York (1998)

Using Hybrid CPU-GPU Platforms to Accelerate the Computation of the Matrix Sign Function

Peter Benner[1], Pablo Ezzatti[2],
Enrique S. Quintana-Ortí[3], and Alfredo Remón[3]

[1] Fakultät für Mathematik, Chemnitz University of Technology,
D-09107 Chemnitz, Germany
benner@mathematik.tu-chemnitz.de

[2] Centro de Cálculo–Instituto de la Computación, Universidad de la República,
11.300–Montevideo, Uruguay
pezzatti@fing.edu.uy

[3] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I,
12.071–Castellón, Spain
{quintana,remon}@icc.uji.es

Abstract. We investigate the numerical computation of the matrix sign function of large-scale dense matrices. This is a common task in various application areas. The main computational work in Newton's iteration for the matrix sign function consits of matrix inversion. Therefore, we investigate the performance of two approaches for matrix inversion based on Gaussian (LU factorization) and Gauss-Jordan eliminations. The target architecture is a current general-purpose multi-core processor connected to a graphics processor. Parallelism is extracted in both processors by linking sequential versions of the codes with multi-threaded implementations of BLAS. Our results on a system with two Intel Quad-Core processors and an NVIDIA Tesla C1060 illustrate the performance and scalability attained by the codes on this system.

Keywords: Matrix sign function, hybrid platforms, GPUs, multi-core processors, linear algebra, high performance computing.

1 Introduction

Consider a matrix $A \in \mathbb{R}^{n \times n}$ with no eigenvalues on the imaginary axis, and let

$$A = T^{-1} \begin{pmatrix} J_- & 0 \\ 0 & J_+ \end{pmatrix} T, \tag{1}$$

be its Jordan decomposition, where the eigenvalues of $J_- \in \mathbb{R}^{j \times j}$ $/J_+ \in \mathbb{R}^{(n-j)\times(n-j)}$ all have negative/positive real parts [1]. The *matrix sign function* of A is then defined as

$$\operatorname{sign}(A) = T^{-1} \begin{pmatrix} -I_j & 0 \\ 0 & I_{n-j} \end{pmatrix} T, \tag{2}$$

where I denotes the identity matrix of the order indicated by the subscript. The matrix sign function is a useful numerical tool for the solution of control theory problems (model reduction, optimal control) [2], and the bottleneck computation in many lattice quantum chromodynamics computations [3] and dense linear algebra computations (block diagonalization, eigenspectrum separation) [1,4]. Large-scale problems as those arising, e.g., in control theory often involve matrices of dimension $n \rightarrow O(10,000 - 100,000)$ [5].

There are simple iterative schemes for the computation of the sign function. Among these, the Newton iteration, given by

$$\begin{aligned} A_0 &:= A, \\ A_{k+1} &:= \tfrac{1}{2}(A_k + A_k^{-1}), \;\; k = 0, 1, 2, \ldots, \end{aligned} \tag{3}$$

is specially appealing for its simplicity, efficiency, parallel performance, and asymptotic quadratic convergence [4,6]. However, even if A is sparse, $\{A_k\}_{k=1,2,\ldots}$ in general are full dense matrices and, thus, the scheme in (3) roughly requires $2n^3$ floating-point arithmetic operations (flops) per iteration.

In the past, large-scale problems have been tackled using message-passing parallel solvers based on the matrix sign function which were then executed on clusters with a moderate number of nodes/processors [7]. The result of this effort was our message-passing library `PLiC` [8] and subsequent libraries for model reduction (`PLiCMR`, see [9]) and optimal control (`PLiCOC`, see [10]). Using this library, 16–32 processors showed to provide enough computational power to solve problems with $n \approx 10,000$ in a few hours.

Following the recent uprise of hardware accelerators, like the graphics processors (GPUs), and the increase in the number of cores of current general-purpose processors, in this paper we evaluate an alternative approach that employs a sequential version of the codes in the `PLiC` library, and extracts all parallelism from tuned multi-threaded implementations of the BLAS (*Basic Linear Algebra Subprograms*) [11,12,13]. The results attained in a hybrid, heterogeneous architecture composed of a general-purpose multi-core processor and a GPU demonstrate that this is a valid platform to deal with large-scale problems which, only a few years ago, would have required a distributed-memory cluster.

The rest of the paper is structured as follows. In Section 2 we elaborate on the hybrid computation of the matrix inverse on a CPU-GPU platform. This is followed by experimental results in Section 3, while concluding remarks and open questions follow in Section 4.

2 High-Performance Matrix Inversion

As equation (3) reveals, the application of Newton's method to the sign function requires, at each iteration, the computation of a matrix inverse. We next review two different methods for the computation of this operation, based on the LU factorization and Gauss-Jordan transformations.

2.1 Matrix Inversion via the LU Factorization

The traditional approach to compute the inverse of a matrix $A \in \mathbb{R}^{n\times n}$ is based on Gaussian elimination (i.e., the LU factorization), and consist of the following three steps:

1. Compute the LU factorization $PA = LU$, where $P \in \mathbb{R}^{n\times n}$ is a permutation matrix, and $L \in \mathbb{R}^{n\times n}$ and $U \in \mathbb{R}^{n\times n}$ are, respectively, unit lower and upper triangular factors [1].
2. Invert the triangular factor $U \rightarrow U^{-1}$.
3. Solve the system $XL = U^{-1}$ for X.
4. Undo the permutations $A^{-1} := XP$.

LAPACK [14] is a high-performance linear algebra library which provides routines that cover the functionality required in the previous steps. In particular, routine `getrf` yields the LU factorization (with partial pivoting) of a nonsingular matrix (Step 1), while routine `getri` computes the inverse matrix of A using the LU factorization obtained by `getrf` (Steps 2–4).

The computational cost of computing a matrix inverse following the previous four steps is $2n^3$ flops. The algorithm sweeps through the matrix four times (one per step) and presents a mild load imbalance, due to the work with the triangular factors.

2.2 Matrix Inversion via Gauss-Jordan Elimination

The Gauss-Jordan elimination algorithm [15] (GJE) for matrix inversion is, in essence, a reordering of the computation performed by matrix inversion methods based on Gaussian elimination, and hence requires the same arithmetic cost.

Figure 1 illustrates a blocked version of the GJE procedure for matrix inversion using the FLAME notation [16,17,18]. There $m(A)$ stands for the number of rows of matrix A. We believe the rest of the notation to be intuitive; for further details, see [16,17]. (A description of the unblocked version, called from inside the blocked one, can be found in [19]; for simplicity, we hide the application of pivoting during the factorization, but details can be found there as well.) The bulk of the computations in the procedure can be cast in terms of the matrix-matrix product, an operation with a high parallelism. Therefore, GJE is a highly appealing method for matrix inversion on emerging architectures like GPUs, where many computational units are available, provided a highly-tuned implementation of the matrix-matrix product is available.

We next introduce three implementations for the GJE method (with partial pivoting) on two parallel architectures: a multi-core CPU architecture and a GPU from NVIDIA. The following variants differ on which part of the computations is performed on the CPU (the general-purpose processor or host), and which part is off-loaded to the hardware accelerator (the GPU or device). They all try to reduce the number of communications between the memory spaces of the host and the device.

```
Algorithm. $[A] := \mathrm{GJE}_{\mathrm{BLK}}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$ and $A_{BR}$ is $n \times n$
while $m(A_{TL}) < m(A)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
        where $A_{11}$ is $b \times b$

    $\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix} := GJE_{\mathrm{UNB}}\left(\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix}\right)$    Unblocked Gauss-Jordan
        $A_{00} := A_{00} + A_{01}A_{10}$    Matrix-matrix product
        $A_{20} := A_{20} + A_{21}A_{10}$    Matrix-matrix product
        $A_{10} := A_{11}A_{10}$             Matrix-matrix product
        $A_{02} := A_{02} + A_{01}A_{12}$    Matrix-matrix product
        $A_{22} := A_{22} + A_{21}A_{12}$    Matrix-matrix product
        $A_{12} := A_{11}A_{12}$             Matrix-matrix product

    Continue with
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
endwhile
```

Fig. 1. Blocked algorithm for matrix inversion via GJE without pivoting

Implementation on a multi-core CPU: GJE(CPU)**.** In this first variant all operations are performed on the CPU. Parallelism is obtained from a multi-threaded implementation of BLAS for general-purpose processors. Since most of the computations are cast in terms of products of matrices, high performance can be expected from this variant.

Implementation on a many-core GPU: GJE(GPU)**.** This is the GPU-analogue to the previous variant. The matrix is first transferred to the device; all computations proceed there next; and the result (the matrix inverse) is finally moved back to the host.

Hybrid implementation: GJE(Hybrid)**.** While most of the operations performed in the GJE algorithm are well suited for the GPU, a few are not. This is the case for fine-grained operations, as the low computational cost and data dependencies deliver low performance on massively parallel architectures like the GPU. To solve this problem, we propose a hybrid implementation. In this new approach, operations are performed in the most convenient device, exploiting the capabilities of both architectures.

In particular, in this variant the matrix is initially transferred to the device. At the beginning of each iteration of the algorithm in Figure 1, the current column panel, composed of $\left[A_{01}^T, A_{11}^T, A_{21}^T\right]^T$ is moved to the CPU and factorized there. The result is immediately transferred back to the device, where all remaining computations (matrix-matrix products) are performed. This pattern is repeated until the full matrix inverse is computed. The inverse is finally transferred from the device memory to the host.

In summary, only the factorization of the current column panel is executed on the CPU, since it involves a reduced number of data (limited by the algorithmic block size), pivoting and BLAS-1 operations which are not well suited for the architecture of the GPU. The matrix-matrix products and pivoting of the columns outside the current column panel are performed on the GPU.

3 Experimental Results

In this section we evaluate four parallel multi-threaded codes to compute the inverse of a matrix:

– LAPACK(CPU): The four steps of the LAPACK approach, with all computations carried out on the CPU and parallelism extracted by using a multi-threaded implementation of BLAS; see subsection 2.1.
– GJE(CPU), GJE(GPU), and GJE(Hybrid): The implementations described in subsection 2.2.

Two different implementations of the BLAS (Goto BLAS [20] –version 1.26– and Intel MKL [21] –version 10.1–) were used to execute operations on the general-purpose processor, while on the NVIDIA GPU, CUBLAS [22] (version 2.1) was the library that we used.

The experiments employ single and double precision and the results always include the cost of data transfers between the host and device memory spaces. The target platform consists of two Intel Xeon QuadCore processors connected to an NVIDIA Tesla C1060. Table 1 offers more details on the hardware.

Figure 2 reports the GFLOPS (10^9 flops per second) rates attained by the different implementations of the inversion codes operating on single-precision matrices with sizes between 1,000 and 8,000. Several algorithmic block sizes

Table 1. Hardware employed in the experiments

Processors	#cores	Frequency (GHz)	L2 cache (MB)	Memory (GB)	Single/Double precision peak performance (GFLOPS)
Intel Xeon QuadCore E5405	8	2.3	12	8	149.1/74.6
Nvidia TESLA c1060	240	1.3	–	4	933.0/78.0

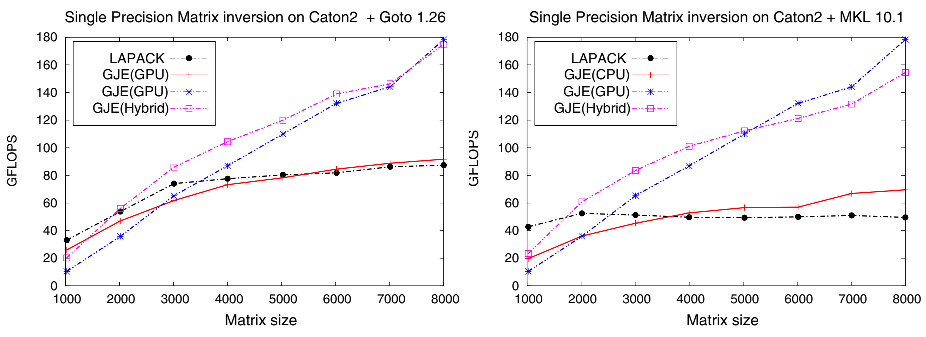


Fig. 2. Performance of the matrix inversion codes

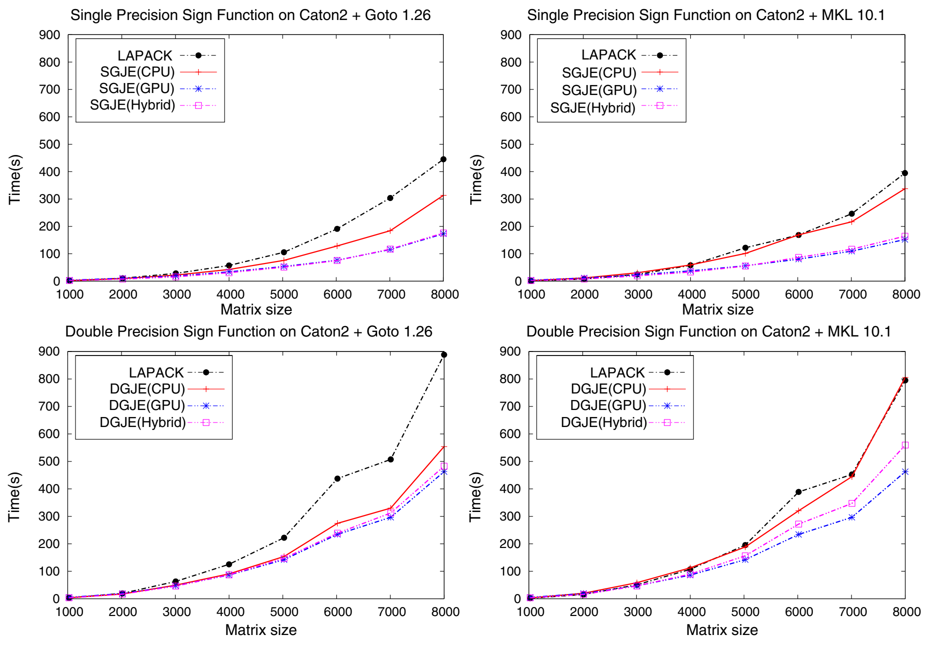


Fig. 3. Execution times of the Newton iteration for the matrix sign function with the matrix inversion implemented using the different variants discussed

(parameter b in Figure 1) were tested but, for simplicity, the results in all figures, hereafter, correspond to those obtained with the optimal block size.

The LAPACK code executed using all 8 cores of the two general-purpose processors yields the lowest GFLOPS rate, while the GJE algorithm using the same resources performs slightly better. Both implementations that employ the GPU outperform the ones executed only on the CPU. The Hybrid approach is the best option for small/medium matrices, while the version executed entirely on the GPU is the best for large matrices.

Figure 3 shows execution times of the Newton's iteration for the matrix sign function, using the previous matrix inversion codes and both single and double precision data. As expected, the LAPACK implementation delivers the highest execution time, followed by GJE(CPU). Codes for GPU are notoriously/slightly faster in single/double precision. Gains from GPU codes are larger for single precision computations and for large matrices.

4 Concluding Remarks and Future Work

We have demonstrated the benefits of using a current GPU to off-load part of the computations in a dense linear algebra operation rich in level-3 BLAS like the matrix inversion. This operation is the basis for the computation of the matrix sign function via Newton's iteration and is also the key to the efficient solution of important problems in control theory such as model reduction and optimal control.

The evaluation of matrix inversion codes clearly identify the superior performance of the procedures based on Gauss-Jordan elimination over Gaussian elimination (the LU factorization).

Our research poses some open questions which form the basis of our ongoing and future work:

– Most applications in control theory and linear algebra require double precision but current GPUs deliver considerable lower performance when they operate with this data type. Is it possible to compute the sign function in single precision and then use iterative refinement [23] to obtain a double precision solution at a low cost?
– Can we overlap CPU and GPU computations so that while the CPU is computing some blocks the GPU updates others? Note that this requires a careful synchronization of the data transfers between the memory spaces of host and device.
– Is it possible to overlap computation on the GPU with data transfers between the CPU and the GPU memory spaces to improve the performance for the small problem sizes?

Acknowledgments

This work was supported by the Spanish Ministry of Science and Innovation/FEDER (contract TIN2008-06570-C04-01), by the Fundación Caixa-Castelló/Bancaixa (contracts no. P1B-2007-19 and P1B-2007-32) and by the Generalitat Valenciana (contract PROMETEO/2009/013).

References

1. Golub, G., Loan, C.V.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, Baltimore (1996)
2. Petkov, P., Christov, N., Konstantinov, M.: Computational Methods for Linear Control Systems. Prentice Hall, Hertfordshire (1991)

3. Frommer, A., Lippert, T., Medeke, B., Schilling, K. (eds.): Numerical Challenges in Lattice Quantum Chromodynamics. Lecture Notes in Computational Science and Engineering, vol. 15. Springer, Heidelberg (2000)
4. Byers, R.: Solving the algebraic Riccati equation with the matrix sign function. Linear Algebra Appl. 85, 267–279 (1987)
5. IMTEK, Oberwolfach model reduction benchmark collection, `http://www.imtek.de/simulation/benchmark/`
6. Benner, P., Quintana-Ortí, E.: Solving stable generalized Lyapunov equations with the matrix sign function. Numer. Algorithms 20, 75–100 (1999)
7. Benner, P., Claver, J., Quintana-Ortí, E.: Parallel distributed solvers for large stable generalized Lyapunov equations. Parallel Processing Letters 9, 147–158 (1999)
8. Benner, P., Quintana-Ortí, E., Quintana-Ortí, G.: A portable subroutine library for solving linear control problems on distributed memory computers. In: Cooperman, G., Jessen, E., Michler, G. (eds.) Workshop on Wide Area Networks and High Performance Computing, Essen (Germany), September 1998. Lecture Notes in Control and Information, pp. 61–88. Springer, Heidelberg (1999)
9. Benner, P., Quintana-Ortí, E., Quintana-Ortí, G.: State-space truncation methods for parallel model reduction of large-scale systems. Parallel Comput. 29, 1701–1722 (2003)
10. Benner, P., Quintana-Ortí, E., Quintana-Ortí, G.: Solving linear-quadratic optimal control problems on parallel computers. Optimization Methods & Software 23, 879–909 (2008)
11. Lawson, C.L., Hanson, R.J., Kincaid, D.R., Krogh, F.T.: Basic linear algebra subprograms for Fortran usage. ACM Trans. Math. Soft. 5, 308–323 (1979)
12. Dongarra, J.J., Croz, J.D., Hammarling, S., Hanson, R.J.: An extended set of FORTRAN basic linear algebra subprograms. ACM Trans. Math. Soft. 14, 1–17 (1988)
13. Dongarra, J.J., Croz, J.D., Hammarling, S., Duff, I.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Soft. 16, 1–17 (1990)
14. Anderson, E., Bai, Z., Demmel, J., Dongarra, J.E., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A.E., Ostrouchov, S., Sorensen, D.: LAPACK Users' Guide. SIAM, Philadelphia (1992)
15. Gerbessiotis, A.V.: Algorithmic and Practical Considerations for Dense Matrix Computations on the BSP Model. PRG-TR 32, Oxford University Computing Laboratory (1997)
16. Gunnels, J.A., Gustavson, F.G., Henry, G.M., van de Geijn, R.A.: FLAME: Formal linear algebra methods environment. ACM Trans. Math. Soft. 27, 422–455 (2001)
17. Bientinesi, P., Gunnels, J.A., Myers, M.E., Quintana-Ortí, E.S., van de Geijn, R.A.: The science of deriving dense linear algebra algorithms. ACM Trans. Math. Soft. 31, 1–26 (2005)
18. University of Texas, `http://www.cs.utexas.edu/~flame/`
19. Quintana-Ortí, E., Quintana-Ortí, G., Sun, X., van de Geijn, R.: A note on parallel matrix inversion. SIAM J. Sci. Comput. 22, 1762–1771 (2001)
20. Texas Advanced Computing Center, `http://www.tacc.utexas.edu/~kgoto/`
21. Intel Corporation, `http://www.intel.com/`
22. Nvidia Corporation, `http://www.nvidia.com/cuda/`
23. Nicholas, J.N.: Accuracy and stability of numerical algorithms, Philadelphia, PA, USA (1996)

Modelling Pilot-Job Applications on Production Grids

Tristan Glatard and Sorina Camarasu-Pop

University of Lyon, CNRS, INSERM, CREATIS, 69621 Villeurbanne, France

Abstract. Pilot-job systems have emerged as a computation paradigm to cope with heterogeneity of production grids, greatly improving fault ratios and latency. Tools like DIANE, WISDOM-II, ToPoS and Condor glideIns are now being widely adopted to conduct large-scale experiments on such platforms. However, a model of pilot-job applications is still lacking, making it difficult to determine submission parameters such as the number of pilots to submit to achieve a given performance level. The variability of production conditions and the heterogeneity of the underlying middleware and infrastructure further complicates this issue. This paper presents a performance model for pilot-job applications running on production grids. Based on a probabilistic modelling, we derive statistics about the number of available pilots along time and the makespan of the application given the number of submitted pilots. Results obtained on a radiotherapy application running on the EGEE production grid show that the model is accurate enough to correctly describe the behavior of the application, setting the basis for further optimization strategies.

1 Introduction

Large-scale production grids such as EGEE[1] are now used by a variety of applications benefiting from computing power and storage space provided by federations of computing centers. Shared by thousands of users, those infrastructures have become complex systems, providing heterogeneity, high variability, low reliability and high latencies as a downside of the huge amount of resources. On EGEE, dozen of minutes of latency with similar standard-deviations and fault ratios of 20% are a common toll to access some of the 80,000 provided CPUs.

Applications relying on such grids had to adopt pragmatic solutions to handle heterogeneity. Among these, pilot jobs provide a submission scheme where tasks are no longer pushed through the grid scheduler but are put in a master pool and pulled by pilots running on computing nodes. Although pilots are still submitted through the regular grid middleware, such a pull model nicely adapts to heterogeneity (pilots running on faster resources pull more tasks than the others), reduces faults and improves latency.

Several pilot-job frameworks have been developed. In particular, systems interfaced with EGEE include DIANE [11], WISDOM-II [1,7], ToPoS[2], BOINC

[1] `http://eu-egee.org/`

[2] `https://wiki.nbic.nl/index.php/ToPoS`

tasks [9] and gPTM3D in radiology [3]. Condor[3] also has its pilot-job submission framework, coined glideIns [15]. However, models are lacking to describe, analyze and optimize the performance of applications relying on pilot-job systems on production grids [8]. This paper proposes such a model and evaluates it in real conditions. Following previous works [5,6], we adopt a probabilistic approach relying on the latency distribution to derive statistics about metrics of interest. The model is detailed in section 2 and evaluated using a radiotherapy application in section 3. Results and capabilities of the model are discussed in section 4.

2 Definitions and Modelling

2.1 Pilot Jobs

Production grids such as EGEE are operated as super-batch systems in which jobs are submitted to a grid scheduler which determines to which computing center they have to be sent. Computing centers then implement their own batch queue to schedule jobs on the computing nodes. By default, applications split the workload into jobs before submitting (i.e. pushing) them to the grid scheduler. Although quite simple, this has disadvantages in terms of fault tolerance and job turn-over. Failed jobs have to be resubmitted to the grid scheduler and go through all middleware components before reaching again a computing node. Besides, heterogeneity can hardly be coped with since the workload is split before the actual computing nodes are known to the application.

In the pilot-job model, the workload is divided in *tasks* by the application and submitted to a *master* managing a task pool on the application host. In parallel, generic grid *jobs* are submitted to the grid scheduler. Once they reach a computing node, those pilots (also called agents or workers) keep on fetching tasks from the master until they all successfully complete. With such a late task-to-node binding, heterogeneity is naturally handled since pilots running on fast resources fetch more tasks than the others. Moreover, failures have a limited impact because failed tasks can directly be executed by other pilots (faulty pilots are removed). Finally, latency is reduced since tasks are directly scheduled by the master on the pilots without going through the whole grid middleware.

2.2 Notations and Assumptions

Given a number of submitted pilots, the modelling aims at estimating (i) the evolution of the number of available pilots along time, i.e., the number of pilots that have reached a computing node and (ii) the makespan of the application, i.e., the duration between the submission of the first pilot and the completion of the last task.

In the following, L denotes the latency (i.e. the total duration between job submission and the beginning of its execution), N the number of available pilots, n the total number of pilots submitted at $t = 0$ and w_0 the total amount of work

[3] http://www.cs.wisc.edu/condor/

to be performed by the application. w_0 is expressed as a work time and includes both computing and data transfers. Probabilistic density functions (*pdf*) are denoted with f and cumulative ones (*cdf*) with F. Capitals are random variables and lowercases are fixed values.

The following assumptions are made. First, pilots are assumed to be submitted at $t = 0$, which assumes that they belong to a specific user and that he/she only executes a single application at a time. Moreover, pilots are also assumed generic, i.e., a given pilot can execute any task of the application.

Besides, failed jobs are considered to have an infinite latency, which is adapted to model pilots that are submitted to the grid but never connect back to the master. Those faults occur with ratio ρ. To improve readability, ρ will be omitted in the following of this section: equations taking faults into account can be derived by replacing F_L by $(1-\rho)F_L$ everywhere. Although this simple model probably does not hold on platforms such as the ones mentioned in [14,2], it is here supported by evidence found in [10] (Fig. 9), based on an analysis of the trace of 33 millions of EGEE jobs submitted between 2005 and 2007. Results show that using such a $(1-\rho)F_L$ fault model instead of the actual fault distribution only has a small impact on the considered parameter estimation.

The modelling also assumes that pilots face independent and identically distributed (*iid*) latencies. In particular, no saturation of the grid scheduler or queues during the application run is considered. This is realistic as long as we consider applications of reasonable size with respect to the grid infrastructure (i.e. the application itself does not cause system saturation, which makes sense on production grids given their large scale) and that no significant burst period occurs during the run. Handling bursts is a problem currently under study and exploiting works such as [12] may further improve our modelling. Another bottleneck potentially breaking this *iid* assumption is the job submission process. Though jobs can usually be submitted at once (e.g., using bag-of-tasks or DAG submission facilities), applications often still use sequential submission for scalability reasons. How the modelling handles this particular case is described in section 2.5. Another potential limit of the *iid* assumption is the evolution of grid conditions while the application is running. In particular, time, day of week and month may have an impact on the latency. To cope with that, experimental data such as the one reported in [4] could be exploited to adapt the model along time.

2.3 Number of Available Pilots

At instant t, the probability to have k pilots available out of n submitted is:

$$f_N(k,t) = \binom{n}{k} F_L(t)^k (1 - F_L(t))^{(n-k)}$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Indeed, any k pilots among n may be available at instant t (i.e. $L \leq t$ for k pilots, which occurs with probability $F_L(t)^k$), while the other $(n-k)$ are still being scheduled or queued ($L \geq t$ for $n-k$ pilots, probability $(1-F_L(t))^{n-k}$). We can notice that at a given instant N follows a binomial distribution with parameter $F_L(t)$. Thus its expectation and variance are:

$$E_N(t) = nF_L(t) \tag{1}$$
$$\sigma_N(t)^2 = n(1 - F_L(t))F_L(t) \tag{2}$$

From equation 1 we have $\lim_{t\to\infty} E_N(t) = n(1-\rho)$, i.e., after enough time, the user will manage to control all resources he/she submits to, up to the failure ratio. This remains realistic as long as the number of submitted pilots is lower than the number of available computing resources, which is reasonable for large-scale grids such as EGEE that gathers some 80,000 CPUs.

2.4 Makespan

We assume here that the total amount of work (CPU time and data transfers) of an application can be split in any number of tasks, and that linear speed-up w.r.t the number of available pilots is achieved. The latter considers that an application running on a given set of pilots would behave as if all the pilots were achieving the average performance. Although asymptotically realistic, this may lead to some inaccuracies in transient phases. To cope with that, the model could be enhanced by including distributions of host performance. However, to our knowledge, no such model is available on EGEE and building it may be challenging since it is very likely to depend on the application.

Let us denote $W(t)$ the remaining work time at time t ($W(0) = w_0$). Since linear speed-up w.r.t the number of pilots is assumed, we have:

$$dW = -N(t)dt$$

so that the remaining amount of work at a given instant is:

$$W(t) = \max(w_0 - \int_0^t N(u)du, 0)$$

and given equation 1:

$$E_W(t) = \max(w_0 - n\int_0^t F_L(u)du, 0)$$

The expectation $E_M(n)$ of the makespan $M(n)$ of an application is then obtained by solving the following in t:

$$w_0 - n\int_0^t F_L(u)du = 0, \quad \text{i.e. :}$$

$$\int_a^{E_M(n)} F_L(u)du = \frac{w_0}{n} \tag{3}$$

where a is the lower bound of the support of F_L (a is the largest value for which $F_L(a) = 0$). In case L is a fixed value (L=a), then $F_L(t) = 1$ for every $t > a$ so that $E_M(n) = a + \frac{w_0}{n}$. We also have the following limit:

$$\lim_{n\to+\infty} E_M(n) = a \tag{4}$$

2.5 Determination of F_L for Sequential Job Submission

In case jobs have to be sequentially submitted to the grid scheduler, the assumption of *iid* latencies obviously does not hold. This is for instance the case when using EGEE's Resource Broker, as done in the experiment section of this paper. In such a case, the submission of a job is delayed by the sum of the submission times of its predecessors, so that the latency faced by the i^{th} submitted job of an application is:

$$R_i = T_i + G$$

where T_i is the submission time of job i (it depends on the number of previously submitted jobs) and G is the rest of the grid latency. On the other hand, grid latency measures (e.g. based on probe monitoring) capture the following:

$$M = S + G$$

where S is the submission time of a single job.

We can consider that T_i are *iid* and the job rank i is uniformly distributed between 1 and n. Thus, $T_i = T$ and T has the following *pdf*:

$$f_T(t) = \frac{1}{n}\left(\sum_{k=1}^{n} f_{kS}(t)\right) = \frac{1}{n}\left(\sum_{k=1}^{n} f_S^{*k}(t)\right)$$

where f_{kS} denotes the *pdf* of random variable $\sum_{j=1}^{k} S$ and f_S^{*k} is f_S convolved k times with itself. Then, $L = T + G$ and we have:

$$F_L(t) = \int_0^{\infty} \frac{1}{n}\left(\sum_{k=1}^{n} f_S^{*k}(u)\right) F_G(t-u)du \qquad (5)$$

Equations 1, 2 and 3 still hold and F_L can be computed from the latency distribution using equation 5.

3 Experiment Set-Up and Results

The modelling was evaluated on a radiotherapy simulation application running on the EGEE grid with the DIANE pilot-job framework [13]. The experiments were conducted on the `biomed` Virtual Organisation (VO) of the EGEE grid, gathering approximately 190 computing sites and more than 40 grid schedulers (Resource Brokers) at the time of the experiment.

Nineteen (19) runs of the application (6h45min each on a 2.4 GHz Intel Core Duo PC) were performed, varying the number of submitted pilots (n) from 25 to 150. Pilots were submitted to a single grid scheduler.

The distribution of the grid latency (F_L) was measured from probe round-trip times, which can be problematic, e.g., when schedulers are configured to use different priorities depending on job length, user or recent resource usage.

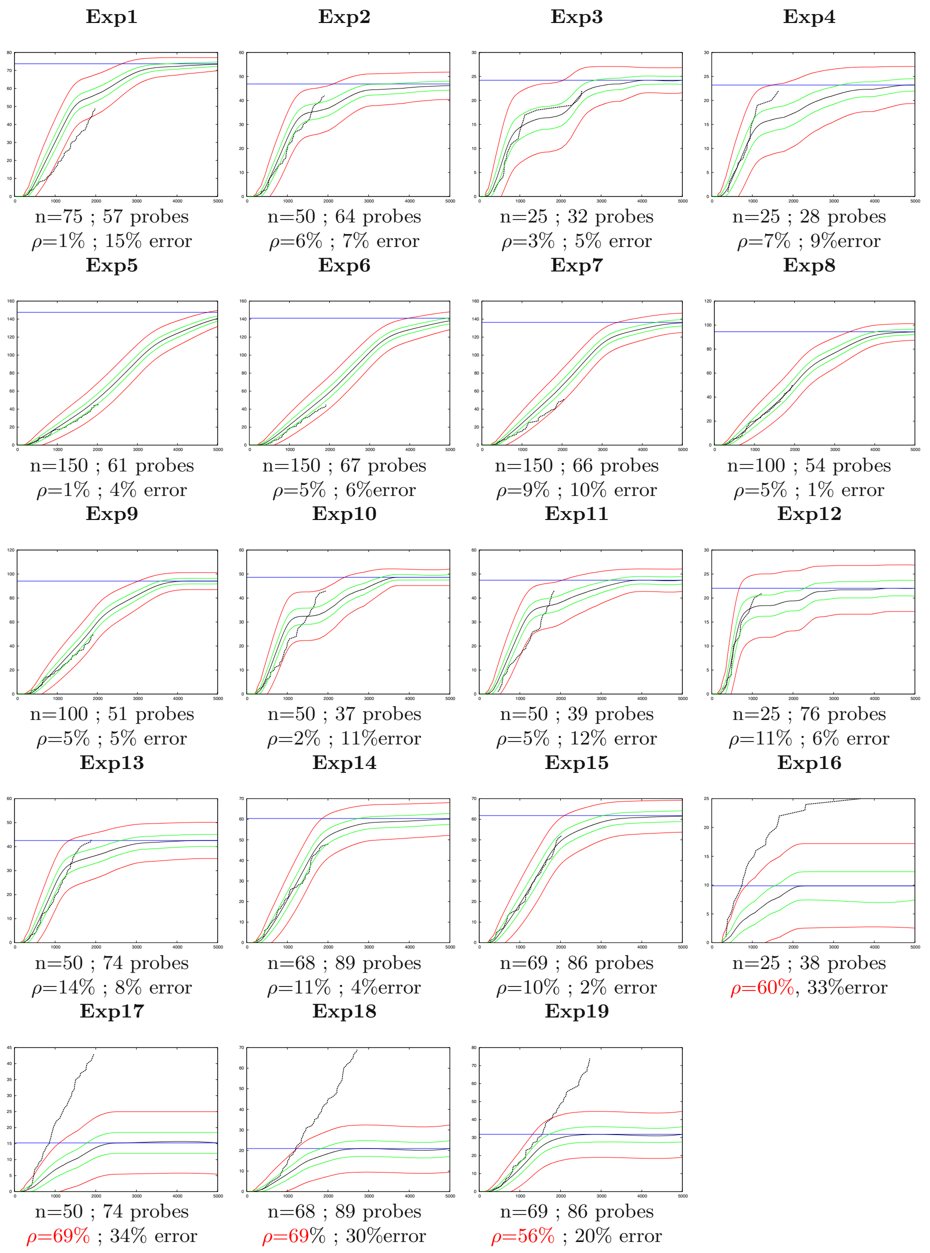


Fig. 1. Number of available pilots along time: the model is figured with plain lines ($E_N(t)$ is the central black line, $\pm\sigma_N(t)$ is in green and $\pm 3\sigma_N(t)$ is in red) and the measure with dashed lines. The x-axis represents time in seconds and the y-axis denotes the number of agents. When monitoring probes are reliable enough (Exp 1 to 15) the model fits the data with a 7% error w.r.t the number of submitted pilots.

In such cases, historical data from the considered user and application could be used to estimate the latency distribution. This could be obtained either by application-level monitoring or by querying middleware services.

To avoid biases, probes and application jobs were submitted from machines located in different cities, using different user credentials and different submission codes. To evaluate the relevance of our modelling of sequential job submission (section 2.5), we used an EGEE Resource Broker with sequential submission.

A fixed number of probes was permanently maintained inside the system. To ensure that enough probes were used to build the latency *cdf*, probes submitted up to 1 hour before the experiment were considered in addition to the ones submitted during the experiment. For each experiment those probes were used to build F_L using equation 5. A total number of 1093 probes has been used for the 19 experiments, among which a global 6.8% fault ratio has been measured. Their mean latency was 767 seconds and the standard-deviation was 766 seconds.

The computation of $E_N(t)$ and $\sigma_N(t)$ was done straight from equations 1 and 2. The total work time w_0 was estimated as the value leading to the minimal mean-square error between modelled and experimental makespans.

3.1 Results

Figure 1 compares the theoretical and measured number of available pilots throughout the 19 experiments. For each graph, the central black plain line corresponds to $E_N(t)$ as computed from equation 1. The interval delimited by plain green lines covers $E_N(t) \pm \sigma_N(t)$ and the red covers $E_N(t) \pm 3\sigma_N(t)$, $\sigma_N(t)$ being computed from equation 2. The blue horizontal line corresponds to $\lim_{n\to\infty} E_N(t)$, i.e. to $n(1-\rho)$ where ρ is the fault ratio among monitoring probes. Experimental data is figured with dashed black lines. For each experiment, the number of submitted pilots n, the number of probes used to parametrize the model, the probes fault ratio ρ and the model error are shown. Model error is computed as $\frac{\frac{1}{k}\sum_{\mathrm{i<k}} |m_i - e_i|}{n}$ with k the total number of measure samples, m_i the value of $E_N(t_i)$ obtained from equation 1, e_i the corresponding experimental value (measure) and n the number of submitted pilots for the experiment.

Figure 2 plots the experimental and modelled makespans. The w_0 value minimizing the mean square error w.r.t the data is 39,600s (11 hours). More than 80 grid sites were used for those experiments.

4 Discussion

4.1 Number of Available Pilots

The model is clearly off on 4 of the 19 experiments presented on figure 1, with mean errors greater than 20% (**Exp 16** to **19**). For those experiments, a high number of faults ($> 55\%$) is observed among the monitoring probes while the application is not impacted. In these cases, the asymptotic line of the model

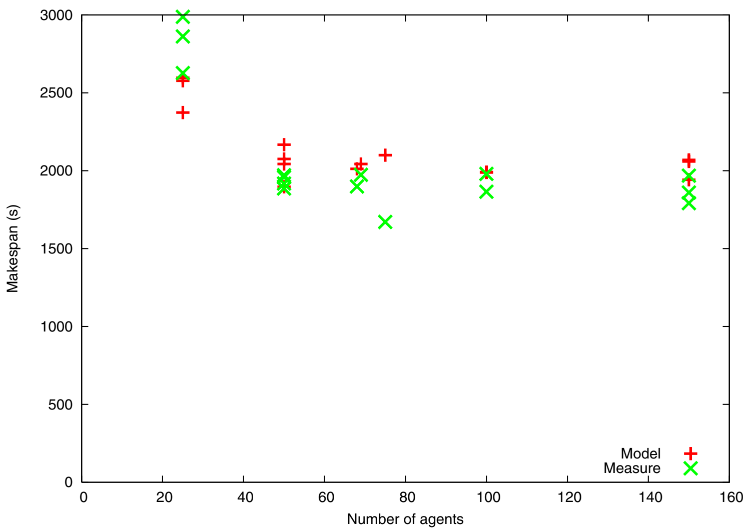


Fig. 2. Measured VS modelled makespans after mean square minimization of the model w.r.t w_0. Mean error is 5min27s; w_0 is 11 hours.

(blue line with y-value $n(1-\rho)$) is very low while the experimental curve grows to regular values. A detailed inspection of the data revealed that such a fault ratio was indeed not faced by pilots submitted during the experiment. Accurately measuring fault ratios is challenging because some kinds of faults occur during a very localized time period, for instance when a service becomes unavailable for a couple of minutes while being restarted. Since the model is parametrized with probes acquired up to 1 hour before the experiment, such localized faults completely puzzle the monitoring system while the experiment may not be impacted at all. In such cases, the grid behavior is too dynamic to be properly captured. In the following discussion we only consider **Exp 1** to **Exp 15** for which fault ratios remain under 15%, a common value on EGEE.

Excluding those 4 experiments, the average model error is 7% with respect to n. This can be considered quite low given the grid variability and the independence of the monitoring probes from the experiments. All the measures but some of **Exp 1** stay in the red interval $[E_N(t)-3\sigma_N(t),E_N(t)+3\sigma_N(t)]$ and a significant proportion is in the green $[E_N(t)-\sigma_N(t),E_N(t)+\sigma_N(t)]$. From a qualitative point of view, the shape of the experimental data is properly captured by the model: for instance, **Exp 3** shows a strong bi-modal behavior nicely described by the model. The model generally tends to overestimate the performance of the application, in particular when the number of submitted pilots is important (see in particular **Exp 1**,**5**, **6**, **7**, **8** and **9** for which $n \geq 75$). This may be due to the fact that in some cases, the makespan is lower than the total submission time, thus reducing the actual number of submitted pilots.

4.2 Makespan Estimation

The experimental makespan (green crosses on figure 2) shows significant variability for a given value of n, which is explained by the variability of grid execution conditions and the heterogeneity of the infrastructure. Still, one can notice a clear decreasing trend between $n = 25$ and $n = 50$ followed by a stable phase around 2000s where the makespan seems to reach a limit, as forecast by the theoretical study in section 2. However, this limit is clearly superior to the value given by the theory, supposed to be a, the lower bound of the support of the *grid* latency *cdf* (150s for the 1093 monitoring probes considered here). The reason for that is obviously the *application* latency, mainly composed of data transfers and the initialization step.

The model (red crosses on figure 2) is able to properly capture the behavior of the experimental makespan. After mean-square error minimization, the mean error between the model and the experimental data is 327 seconds (5 min 27 s). Given the standard-deviation measured on the latency of the monitoring probes (766s), this error can be considered as very small. The dynamicity of the grid is captured thanks to the monitoring system while the evolution with respect to n is correctly described by the probabilistic model.

Finally, the w_0 value given by the minimization (11 hours) approximates the mean measured w_0 (11 hours 13min) with an error of only 2%, which is extremely accurate for an application running in production conditions. We thus conclude that (i) the model is able to explain the observed makespan and (ii) the model is able to estimate the total work time w_0 of the application.

5 Conclusion

We presented a model for pilot-job applications, an execution scheme becoming increasingly adopted by applications to cope with heterogeneity of production grids. Based on the distribution of the latency, the mean and standard-deviation of the number of available pilots along time as well as the makespan of the application are described. An estimation of the total work time of the application is also provided. Sequentially submitted jobs can also be handled. Faults are taken into account by applying a basic transformation on the latency distribution.

Nineteen (19) experiments have been carried-out on a radiotherapy simulation application running on the EGEE production grid using DIANE pilot jobs. Relying on a parametrization of the model using independent monitoring probe jobs, results show that (i) the model is able to correctly describe the evolution of the number of available pilots along time, provided that the fault ratio of the monitoring probes remains reasonable, (ii) the makespan of the application is accurately described and (iii) the total work time of the application is estimated with a 2% error.

The proposed model is thus a suitable tool for analyzing the behavior of pilot-job applications. Though the experiments were carried-out using DIANE, the modelling does not include any implementation-specific assumption, which makes the results applicable to other pilot jobs systems deployed on the same type of production grid infrastructures.

Acknowledgement

We warmly thank David Sarrut for the expertise provided with the radiotherapy application, Jakub Mosciki for his assistance with the DIANE framework and Thomas Grenier for fruitful discussions. We are also grateful to the EGEE European project for providing the grid infrastructure and related user support.

References

1. Ahn, S., Namgyu, K., Seehoon, L., Soonwook, H., Dukyun, N., Koblitz, B., Breton, V., Sangyong, H.: Improvement of Task Retrieval Performance Using AMGA in a Large-Scale Virtual Screening. In: NCM'08, pp. 456–463 (September 2008)
2. Fu, S., Xu, C.-Z.: Exploring event correlation for failure prediction in coalition of clusters. In: Supercomputing (2007)
3. Germain, C., Loomis, C., Mosciki, J.T., Texier, R.: Scheduling for Responsive Grids. JGC 6(1), 15–27 (2008)
4. Glatard, T., Lingrand, D., Montagnat, J., Riveill, M.: Impact of the execution context on Grid job performances. In: WCAMG'07 (CCGrid'07), pp. 713–718 (May 2007)
5. Glatard, T., Montagnat, J., Pennec, X.: Probabilistic and dynamic optimization of job partitioning on a grid infrastructure. In: PDP'06, pp. 231–238 (February 2006)
6. Glatard, T., Montagnat, J., Pennec, X.: Optimizing jobs timeouts on clusters and production grids. In: CCGrid'07, pp. 100–107 (May 2007)
7. Jacq, N., Salzeman, J., Jacq, F., Legre, Y., Medernach, E., Montagnat, J., Maass, J., Reichstadt, M., Schwichtenberg, H., Sridhar, M., Kasam, V., Zimmermann, M., Hofmann, M., Breton, V.: Grid-enabled Virtual Screening against malaria. JGC 6, 29–43 (2008)
8. Juve, G., Deelman, E.: Resource Provisioning Options for Large-Scale Scientific Workflows. In: eScience'08, pp. 608–613 (December 2008)
9. Kacsuk, P., Farkas, Z., Fedak, G.: Towards making BOINC and EGEE interoperable. In: eScience'08, pp. 478–484 (December 2008)
10. Lingrand, D., Montagnat, J., Martyniak, J., Colling, D.: Analyzing the EGEE production grid workload: Application to jobs submission optimization. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2009. LNCS, vol. 5798, pp. 37–58. Springer, Heidelberg (2009)
11. Mosciki, J.T.: Distributed analysis environment for HEP and interdisciplinary applications. Nuclear Instruments and Methods in Physics Research A 502, 426–429 (2003)
12. Ngoc Minh, T., Wolters, L.: Modeling Job Arrival Process with Long Range Dependence and Burstiness Characteristics. In: CCGrid'09, pp. 324–330 (May 2009)
13. Sarrut, D., Guigues, L.: Region-oriented ct image representation for reducing computing time of monte carlo simulations. Med. Phys. 35(4) (2008)
14. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: DSN, pp. 249–258 (2006)
15. Sfiligoi, I.: glideInWMS: a generic pilot-based workload management system. Journal of Physics: Conference Series 119(6) (2008)

Modeling Resubmission in Unreliable Grids: The Bottom-Up Approach

Vandy Berten[1] and Emmanuel Jeannot[2]

[1] Université Libre de Bruxelles
[2] INRIA, LORIA

Abstract. Failure is an ordinary characteristic of large-scale distributed environments. Resubmission is a general strategy employed to cope with failures in grids. Here, we analytically and experimentally study resubmission in the case of random brokering (jobs are dispatched to a computing elements with a probability proportional to its computing power). We compare two cases when jobs are resubmitted to the broker or to the computing element. Results show that resubmit to the broker is a better strategy. Our approach is different from most existing race-based one as it is a bottom-up one: we start from a simple model of a grid and derive its characteristics.

1 Introduction

Computational grids such as EGEE [6] or TeraGrid [13] are routinely used for executing scientific jobs. However, such environment are subject to failures. Indeed, a 9 months study [5] (from Feb. 2006 to Nov. 2006) of the SEE virtual organization of EGEE shows that only 30% of jobs finished with an `OK` status at the first try and 10% ultimately failed. Another study [10] covers the submission of 230 474 jobs in the the EGEE/LCG Grid during 280 days, among which 23 208 (9.93%) failed.

In order to tackle reliability problem, the main strategy commonly employed is to resubmit a failed job [3,1,8]. There are several ways to resubmit a job. It is possible to resubmit the job to the scheduler or to the computing element allocated to it.

To the best of our knowledge, comparing these two strategies in the context of computational grids has never been rigorously conducted. The goal of this work is therefore to model these strategies and to provide experimental insights on when and how a strategy is better than the other. We focus on a special kind of a grid environment directly inspired from EGEE. Jobs are submitted to a resource broker that dispatches them to a computing element where they are queued and treated according to their arrival date.

In general, scheduling algorithms that allocate jobs to resources assume that the arriving time and/or the duration of the jobs are known in advance. However, such an assumption is not always realistic (the duration of the job can only be known after its execution, and the arrival depends on the clients hence is not always deterministic). To cope with this uncertainty, we use a stochastic model where the job inter-arrival follows a Poisson law and the job duration follows an exponential law. Therefore, the resource brokering algorithm is a random one where each job is allocated to a computing element with a probability proportional to its accumulated speed. Thanks to this approach, very few assumptions are required, which leads to a general case that could easily be applied to production environments.

Moreover, to model the unreliability of the environment, we assume that each node has a probability of failure (*i.e* the probability that a job is not correctly executed on this node). Furthermore, to increase realism, we do not assume that this characteristic is known by the resource broker.

The contribution of our paper is the following. First, we model the behavior of the resubmission strategies. Second, we assess the quality of these models by comparing the predicted values with the real ones. We show that in almost every case our models are very precise. Last, we compare the strategies with regards to the execution time of a given set of jobs and we are able to determine when it is better to use a given strategy. It is important to note that our methodology is bottom-up one while most of the resent studies are top-down. This means that in this paper we start from a simple model of a grid and try to analytically compute the model while in the top-down approach, the model is derived from traces.

2 Related Work

Modeling a grid system (waiting time, throughput, failure) has already been studied in the literature. Concerning job duration and submission reference studies are [10] concerning EGEE, or [7,9] concerning different traces. Concerning modeling failures [12] covers high performance computing systems and [11], study the Condor system.

The common methodology of all these approaches is that they tackle the problem using a top-down approach. The top down-approach (or trace-based approach) consists in deriving the characteristics through an analysis of the traces. This approach is very useful to derive accurate understanding of a given setting but is limited when one change the target environment. In our approach (bottom-up) we start from a very general model of the environment and compute analytically a model of the behavior of this environment. This study is therefore an attempt to bridge the gap between the two approaches (trace-based/top-down vs. analytical/Bottom-up).

3 System, Job, and Resubmission Models

3.1 Grid Model

The model of computational grid we propose here is directly inspired from existing ones such as EGEE [6] and has already been presented in [2]. This system is composed of $\mathcal{N}$ sites (or computing elements) $\mathbb{C}_i$. Each site $\mathbb{C}_i$ is assumed to be homogeneous: it is composed of c_i homogeneous computing nodes of speed s_i. The *power* of a computing element is the product of the number of nodes times the speed of its node: $c_i \times s_i$.

A set of jobs (or tasks) is submitted to this grid. All jobs, as most of those submitted to EGEE, are sequential (*i.e.* executed by a single processor).

A job is submitted to a *resource broker* or *job dispatcher*. The broker uses a random strategy to dispatch jobs to the computing elements. Computing element $\mathbb{C}_i$ has a probability $\gamma_i = \frac{c_i \times s_i}{C}$ to be chosen, where $C = \sum_{i=1}^{\mathcal{N}} c_i \times s_i$ is the total power of the considered grid system. Therefore, jobs are submitted to a computing element with a probability proportional to its power. The intuition beyond this strategy is that the more powerful a computing element, the more jobs should be submitted to it. Moreover, it is important to note that this strategy is very simple to implement and does not require to estimate job duration.

We use a stochastic model for the submission date of jobs and their execution duration. The arrival of jobs to the resource broker follows a poissonian distribution with an inter-arrival rate λ. In this paper we focus on computational grids and therefore, we only consider coarse-grain jobs, where the run-time is dominating the whole execution time (*i.e.* network latencies and transfer times are neglected and we thus do not take into account the data localization). Hence, each submitted job l is given a number of operations to perform μ_ℓ and the execution time on computing element $\mathbb{C}_i$ is $\mu_\ell \times s_i$. The only assumption made about μ_ℓ is that it follows an exponential distribution of parameter μ. Each computing element is equipped with a FIFO queue where incoming jobs are stored and wait for a node to become available for execution.

A grid system is never perfectly reliable: job can fail due to several factors. To model that, each node of site $\mathbb{C}_i$ has a probability $p_i < 1$ to produce a faulty result. As computing this probability is not always easy or possible, the resource broker does not use this parameter when dispatching jobs to resources. Finally, we suppose that failures are transient: when a job is erroneous it is possible to use again the processors that has executed this job.

3.2 Fault-Tolerant Strategies

In order to cope with failures, we propose 2 fault tolerant strategies. **Global resubmission**: when a job has failed, it is resubmitted to the resource broker. The resource broker then chooses a new computing element using the same strategy as described above. **Local resubmission**: when a job has failed on a given computing element, it is resubmitted and queued to the same computing element.

We will consider two variants. One with a limited number of R submissions, the other with an unbounded number of resubmissions ($R = +\infty$). The system stops resubmission when the job is correct or after R submissions. When R is finite, it is not guaranteed that the job will finally be correctly executed, while when R is unbounded, it can take a very long time for the job to be correctly executed.

4 Analysis of the Strategies

We propose to study 3 metrics. The *saturation load* which is the load above which the system cannot treat all the incoming jobs (the queue sizes are growing with time). The *average waiting time* which is the time during which a job stays in the system. The last metric is the *failure probability* which gives the chance that this job is not correctly executed.

4.1 Unlimited Global Resubmission

In this section, we assume that a job can be resubmitted an unlimited number of times. This guarantees that, when it exits the system, a job is correct.

Saturation Load. Let $\nu = \frac{\lambda}{\mu C}$ be the input load (the input rate divided by the computational rate). Let $\alpha = \frac{1}{C}\sum_{i=1}^{N} c_i p_i s_i = \sum_{i=1}^{N} \gamma_i p_i$ be the probability that a job scheduled by the Resource Broker fails. Let δ_i be what exits $\mathbb{C}_i$. A part of this flow is sent back to the resource broker, and is then added to the input rate λ. We assume that the input flow plus the resubmitted jobs can be considered as a poissonian flow (this is an approximation, but we assume that if the input flow is (much) larger than the feedback, this assumption is reasonable). We denote by λ_E the effective input rate, *i.e.* λ plus the resubmissions.

We are here interested by non saturated systems ($\frac{\lambda_E}{\mu C} < 1$), which means that the input flow is equal to the output flow. Indeed, the system does not saturate when the output rate λ_E is lower than the computational rate μC. With this approximation, we have δ_i(output rate of $\mathbb{C}_i$) $= \lambda_E \frac{c_i s_i}{C}$(input rate of $\mathbb{C}_i$) and $\lambda_E = \lambda + \sum_k \delta_k p_k = \lambda + \sum_k \lambda_E \frac{c_k s_k}{C} p_k = \lambda + \lambda_E \frac{1}{C}\sum_k c_k p_k s_k = \lambda + \lambda_E \alpha = \frac{\lambda}{1-\alpha}$.

Therefore, if the system input flow is $\nu = \frac{\lambda}{\mu C}$, we have an effective load of $\nu_E = \frac{\lambda_E}{\mu C} = \frac{\frac{\lambda}{1-\alpha}}{\mu C} = \frac{\nu}{1-\alpha}$. Hence, the system saturates for an input load of $\nu = \frac{\lambda}{\mu C} > 1 - \alpha$.

Average traversal time of $\mathbb{C}_i$ (F_i). For a load of ν_E, we know [2] that the average queue size of computing element $\mathbb{C}_i$ is $Q_i(\nu_E) = \mathbb{E}[Q_i] = b\frac{{\nu_E}^{c_i+1} \cdot c_i^{c_i}}{c_i!(1-\nu_E)^2}$ where $b = \left[\sum_{k=0}^{c_i-1} \frac{(\nu_E c_i)^k}{k!} + \frac{(\nu_E c_i)^{c_i}}{c_i!}\frac{1}{1-\nu_E}\right]^{-1}$. On computing element $\mathbb{C}_i$, the average time before the end of an execution (called F_i), with an effective load of ν_E on $\mathbb{C}_i$ is then $\frac{Q_i(\nu_E)}{\mu_i c_i} + \mu_i^{-1}$. Hence, if the input load is ν, $F_i(\nu) = \frac{Q_i(\frac{\nu}{1-\alpha})+c_i}{\mu_i c_i}$. Let $\mathcal{F}$ be the average traversal time. We then have $\mathcal{F} = \sum_{i=1}^{N} \gamma_i F_i$, because γ_i is the probability for a job to be sent on $\mathbb{C}_i$. Hence, $\mathcal{F} = \frac{1}{C}\sum_{i=1}^{N} c_i s_i F_i = \frac{1}{C}\sum_{i=1}^{N} \frac{c_i s_i}{\mu_i c_i}(Q_i(\frac{\nu}{1-\alpha}) + c_i) = \frac{1}{\mu C}\left[\sum_{i=1}^{N}\left(Q_i(\frac{\nu}{1-\alpha}) + c_i\right)\right]$.

Average waiting time. Let W be the average time before a job exits the system, and w_i the average waiting time for a job starting running on $\mathbb{C}_i$ (whatever the number of rounds). We have: $W = \sum_{i=1}^{N} \gamma_i w_i$ because γ_i is the probability for a job to be sent on $\mathbb{C}_i$, and $w_i = F_i + p_i W$ because the average waiting time can be split in the average waiting time on $\mathbb{C}_i$ (F_i), plus, if the job failed (with a probability p_i), the same waiting time than a job entering the system (W). We then have $W = \sum_{i=1}^{N} \gamma_i F_i + \sum_{i=1}^{N} \gamma_i p_i W = \mathcal{F} + \alpha W = \frac{1}{1-\alpha}\mathcal{F}$.

4.2 Limited Global Resubmission

In this section, we add the constraint that a job cannot be executed more times than some value R given by the system.

Saturation load. We first compute the probability for a job to fail. If a job failed, it means that it failed at each rounds, included the R^{th}. We then have: $\mathbb{P}[\text{failure}] = \mathbb{P}[\text{failure at the first round} \wedge \cdots \wedge \text{failure at the } R^{\text{th}} \text{ round}] = \prod_{i=1}^{R} \mathbb{P}[\text{failure at the } i^{\text{th}} \text{ round}] = \prod_{i=1}^{R} \alpha = \alpha^R$. The probability for a job to exit the system at the end of its execution on $\mathbb{C}_i$ is the probability either to succeed ($1-p_i$), or to have already $R-1$ failures before the current round ($\alpha^{R-1}p_i$). The proportion of jobs exiting $\mathbb{C}_i$ but sent back to the input is then $p_i(1-\alpha^{R-1})$. We can now compute the input flow: $\lambda_E = \lambda + \sum_{k=1}^{N} \delta_k p_k (1-\alpha^{R-1}) = \lambda + (1-\alpha^{R-1}) \sum_{k=1}^{N} \lambda_E \gamma_k p_k = \lambda + (1-\alpha^{R-1})\lambda_E \alpha = \frac{\lambda}{1-(1-\alpha^{R-1})\alpha} = \frac{\lambda}{1-\alpha+\alpha^R}$.

Therefore, we can show that if the input load is ν, we have an effective load $\nu_{_E}$ of $\frac{\nu}{1-\alpha+\alpha^R}$. According to this, the system will then saturate at a load of $1-\alpha+\alpha^R$.

Average traversal time on $\mathbb{C}_i$ (F_i). With the same kind of arguments and notation as for the limited resubmission, we have $F_i = \frac{Q_i(\nu_E)}{\mu_i c_i} + \mu_i^{-1}$. Therefore, if the input load is ν, $F_i(\nu) = \frac{Q_i\left(\frac{\nu}{1-\alpha+\alpha^R}\right)}{\mu_i c_i} + \mu_i^{-1}$. We then have $\mathcal{F} = \frac{1}{\mu C}\left[\sum_{i=1}^{N}\left(Q_i\left(\frac{\nu}{1-\alpha+\alpha^R}\right) + c_i\right)\right]$.

Waiting time. We will use the following notations: W_k is the average time that a job entering the system takes to exit the system, if it can still be resubmitted k times (W_0 is then the average run-time of a job which exits the system at the end of its execution, even if it fails). $w_{k,i}$ is the average time a job entering $\mathbb{C}_i$ takes to exit the system, if it can still be resubmitted k times. Hence, when R submissions are allowed W_{R-1} is the average time for executing a job.

Lemma 1. *$W_{R-1} = \frac{1-\alpha^R}{1-\alpha}\mathcal{F}$ where $\mathcal{F}$ is defined in the above section.*

Due to lack of space, the proof is not given. Note that we have $\lim_{k\to\infty} W_k = W$, where W is the average waiting time for unlimited resubmission.

4.3 Unlimited Local Resubmission

In this scenario, a failed job is re-submitted in the queue of the computing element where it was running. We use the same notation ($\lambda_i = \lambda\gamma_i$, λ_i' and δ_i) as in Section 4.1.

Saturation load. In order to evaluate the saturation load, we assume that the feedback flow ($\delta_i p_i$) is poissonian. We have $\lambda_i' = \delta_i$ (the flow coming in the queue equals the flow going out), and $\lambda_i + \delta_i p_i = \delta_i$. Therefore, $\lambda'_i = \delta_i = \dfrac{\lambda_i}{1-p_i}$. The saturation load of $\mathbb{C}_i$ is then $1 - p_i$. And the saturation load of the system is then $\min(1-p_i) = 1 - \max p_i$.

Average traversal time on $\mathbb{C}_i$ (F_i). With the same argument as for the global resubmission, we get $F_i(\nu) = \frac{Q_i(\frac{\nu}{1-p_i})+c_i}{\mu_i c_i}$.

We can see that the traversal time of the queue i is only influenced by its own failure probability, while in the global case, this traversal time depends upon every p_k (in α).

Average waiting time. We use the W and w_i definition from global resubmission. We have $W = \sum_{i=1}^{N} \gamma_i w_i$ and $w_i = F_i + p_i w_i = \frac{F_i}{1-p_i}$.

Therefore, $W = \sum_{i=1}^{N} \gamma_i \frac{F_i}{1-p_i} = \sum_{i=1}^{N} \frac{c_i s_i}{C} \frac{Q_i(\frac{\nu}{1-p_i})+c_i}{\mu s_i c_i (1-p_i)} =$

$$\frac{1}{\mu C} \sum_{i=1}^{N} \frac{1}{1-p_i} \left(Q_i(\tfrac{\nu}{1-p_i}) + c_i \right).$$

4.4 Limited Local Resubmission

We can use the same argument as in the global case: the failure probability on $\mathbb{C}_i$ is ${p_i}^R$. The failure probability for a job entering the system is $\sum_{i=1}^{N} \gamma_i {p_i}^R$.

Effective load. The probability that a job exits the system is the probability that it was correct, or had already too many rounds: $\mathbb{P}[\text{resubmission}] = 1 - \mathbb{P}[\text{exit}] = 1 - \mathbb{P}[\text{succeed} \vee \text{final failure}] = 1 - ((1-p_i) + {p_i}^R) = p_i(1-{p_i}^{R-1})$.

Then the input flow of $\mathbb{C}_i$ is $\lambda'_i = \lambda_i + \delta_i p_i (1-{p_i}^{R-1}) = \lambda_i + \lambda'_i p_i (1-{p_i}^{R-1}) = \frac{\lambda_i}{1-p_i+{p_i}^R}$.

Saturation load. We can then get that the saturation load on $\mathbb{C}_i$ is $1-p_i+{p_i}^R$. Then, the system one is $\min(1-p_i+{p_i}^R) = 1 - \max(p_i - p_i^R)$.

For the Average traversal time of $\mathbb{C}_i$ (F_i), it is straightforward that $F_i(\nu) = \frac{Q_i\left(\frac{\nu}{1-p_i+{p_i}^R}\right)+c_i}{\mu_i c_i}$.

Average waiting time

Lemma 2. *Using the same notation as for the global resubmission we have:* $W_{R-1} = \frac{1}{\mu C} \sum_{i=1}^{N} \frac{1-{p_i}^R}{1-p_i} \left(Q_i \left(\frac{\nu}{1-p_i+{p_i}^R} \right) + c_i \right)$.

Due to lack of space, the proof is not given. Here again we have $\lim_{k\to\infty} W_k = W$, where W is the average waiting time for unlimited local resubmission.

5 Experimental Validation

5.1 Experimental Settings

We have used the SimGRID simulator [4] to perform a set of experiments for measuring the different metrics proposed in the above sections as well as the total running time for the execution of a given number of jobs. For these simulations, the unit of time is set as the average job run-time and hence $\mu = 1$. We have designed 6 platforms where we have executed 1000, 4000, 7000 and 10000 jobs. A sum-up of the six platforms is shown in Table 1. Platforms characteristics vary in number of clusters, number of nodes per cluster, speed of the nodes and failure probability of each nodes. The load submitted to each platform (λ) is set such as saturated and non-saturated modes are both observed. For each platform we use three *failure probability factors* of 1, 0.1 and 0.01. This is a multiplication factor that is applied to the p_i value given in Table 1. The goal of these factors to obtain 3 different variants from low reliability (probability factor set to 0.1) to high reliability (probability factor set to 1). For instance, with a probability factor of 0.1 the probability failure of the 4 computing elements of platform 4 switches from (0.7,0.5,0.3,0.1) to (0.07,0.05,0.03,0.01)).

Table 1. Description of the different platforms used for the experiments

Platform id	Nb clusters	Nodes per cluster c_i	Node speed s_i	Node proba of failure: p_i	λ min:inc:max
1	3	(3,2,5)	(1,3,2)	(0.5,0.5,0.5)	1:0.5:15
2	6	(20,30,10,30,3,50)	(0.1,0.1,0.2,0.1,2,0.05)	(0.5,0.4,0.5,0.2,0.9,0.1)	1:0.5:25
3	4	(6,6,6,6)	(0.1,0.3,0.5,0.7)	(0.1,0.3,0.5,0.7)	1:0.5:25
4	4	(6,6,6,6)	(0.1,0.3,0.5,0.7)	(0.7,0.5,0.3,0.1)	1:0.5:25
5	4	(6,6,6,6)	(0.1,0.3,0.5,0.7)	(0.9,0.9,0.01,0.01)	1:0.5:25
6	4	(6,6,6,6)	(0.4,0.4,0.4,0.4)	(0.99,0.99,0.01,0.01)	1:0.5:25

We have set R, the maximum number of submission from 1 (no resubmission) to 10 and to infinity. The experiments were done by executing each setting (number of jobs, platform id, different load and 3 probability factors) 50 times to obtain relevant average values. At the end, a total of more than 5 millions experiments have been run.

5.2 Model Validation

For each experiments, We have measured the different metrics. Here, we compare our models with the experimental measures to assess their quality.

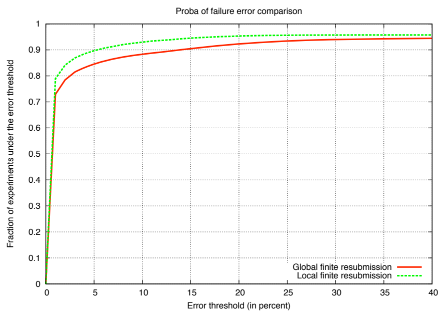
Saturation. When a system is not saturated, the measured average waiting time of job does not depend on the number of submitted jobs. However, when a system is saturated, the input load exceeds its treatment capacities, therefore, the average waiting time increases with the number of submitted job.

For each tuple (probability factor, heuristic, platform, number of copies/ maximum resubmission, λ) we have measured how the average queue size varies when the number of job increases from 1000 to 10000. This increase is measured by the slope of a linear interpolation. When this slope is smaller than a threshold, there is no saturation and when this slope is greater than a threshold there is saturation. By binary search we have found that the best discriminating threshold is about 0.001 (for instance the average waiting time is 10 for a 1000 jobs and 19 for 10000 jobs). A good news is that this value does not depend on the considered heuristic (local resubmission or global resubmission). More precisely, for global resubmission, only 0.14% of the cases have a slope greater than 0.001 while considered by the model as non saturated cases and 0.7% have a slope lower than 0.001 while considered by the model as saturated cases. For local resubmission the percentage of error are respectively 0.6% and 3.9%.

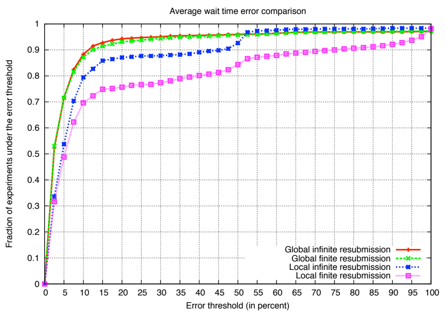
From the above results, we see that models for resubmission are very accurate (local resubmission being a little bit less accurate due to some simplification hypothesis made).

Probability of failure. The accuracy of our model for determining the probability of failure is shown in Figure 1(a). For each case and each heuristic, we have compared the probability of failure given by the model ($pf1$) and the fraction of failure we have measured during our simulation ($pf2$). The error e of the model is computed as follows $e = 100\frac{|pf2-pf1|}{pf2}$. In the graph of Fig. 1(a), we plot an error threshold on the x-axis and the fraction of cases that have an error lower than this threshold on the y-axis. Hence, such graph is similar to a cumulative distribution function (CDF) as for any value on the x-axis, we plot the fraction of experiments that have an error lower than a given value.

Note that unlimited resubmission is not shown here as the failure probability is 0. From this figure we see that more than 80% of the cases have an error lower than 5%. We see that the model for global resubmission is the least accurate though still very good (more than 90% of the cases have an error lower than 15%).



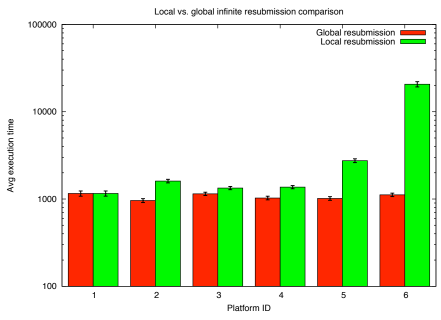
(a) Failure probability



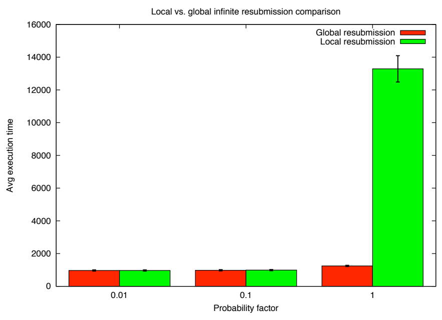
(b) Average wait time

Fig. 1. Accuracy of the model

Average waiting time. To show the accuracy of the model concerning average waiting time, we show the same kind of graph as above: in Fig. 1(b), we plot the error threshold on the x-axis and, on the y-axis, the fraction of experiments that have an average waiting time error lower than this fraction. The error being computed as $e = 100\frac{|wt2-wt1|}{wt2}$, where $wt1$ is the model prediction and $wt2$ is the measured value.



(a) Platform view



(b) Probability factor view

Fig. 2. Comparison of the unlimited local vs. global resubmission

We see that the global resubmission model (both for the unlimited or limited case) is very accurate, almost 90% of the experiments have an error lower than 10%. The local resubmission model is less accurate but the unlimited case is still very good. The model for the local limited resubmission is acceptable with half of the cases having an error lower than 5% and 80% of the cases having an error lower than 40%.

5.3 Comparison of the Heuristics

Local vs. Global Unlimited Resubmission. Here we compare the unlimited resubmission heuristics (jobs are resubmitted to the system until they are successfully executed). Therefore, the probability of success is 1.

In Fig. 2(a) we show the average run-time of both heuristic for the 6 different considered platforms. We see that the local strategy never outperforms the global one. The two strategies provide similar results when no computing element is highly unreliable (platform 1 and to a less extent platforms 3 and 4). When the highest unreliable computing element has a probability of failure close to 0.9 (platform 2 and 5) the global strategy is better than the local one. When a platform has a highly unreliable computing element (such as in platform 6) the global heuristic greatly outperforms the local one.

In Fig. 2(b) we show the average run-time of both strategies when varying the *probability factor*, which is multiplicative factor of the given probability of failure of each computing element of the platform. From that figure, we see that when the probability factor is low (*i.e.* platforms are reliable), the local strategy matches the global one. However, when the probability of failure increases (probability factor of 1), the local strategy is outperformed by an average factor of 10.

We see that the local resubmission outperforms the global resubmission in 2.6% of the cases when the probability factor is 1 (platform are highly unreliable), in 36.6% of the cases when the probability factor is 0.1 and 49.1% of the cases when the probability factor is 0.01 (the platform is fairly reliable). This confirms that the less reliable the platform the less efficient the local resubmission. In any cases, the local resubmission never outperforms the global resubmission with a ratio better than 1.08. This means that when local resubmission is better than global resubmission this is only by a very small margin.

Global resubmission is better than the local resubmission is explained by the fact that, if a job fails, this means that the processor that has executed this job is not reliable and therefore it is better to not reuse it for further execution and hence, it is better to resubmit the job globally to the system.

Local vs. Global limited Resubmission. Here, we compare both approaches when only limited resubmission is possible. When the maximum number of resubmission is allowed, the probability of success is lower than 1 and is different if one use the local or global submission. We already know that when probability of success is one (unlimited resubmission), the global strategy is better than the local strategy. Hence, here, we fix the maximum probability of failure and see if the local strategy can, sometimes outperform the global one. To do so, for each possible combination (platform, input load, and probability factor), we have fixed the success threshold to the one obtained by the global strategy when *one resubmission* is allowed (*i.e.* $R = 2$ and the exact success probability is α^2). From Table 2, we see that, in more than 90% of the cases the local resubmission needs more than 2 resubmissions to exceed this success threshold of the global resubmission when $R = 2$. Hence, the local resubmission needs more resubmission to achieve the same reliability. Moreover, when the success threshold is exceeded by the local strategy, it leads to a better throughput in only 31% of the cases. But, in these cases, when we compare the throughput of the local strategy to the one of the global strategy we see that the improvement is only marginal (at most 4.3%) (detailed results for all platforms are given in table 2). This means that the local strategy sometimes requires a lot of resubmissions to match the reliability of the global strategy. When it does so, it is at the cost of a lower throughput in general and in any case, the local strategy is almost never able to outperform both the reliability and the throughput of the global strategy. Our explanation comes from the fact that, when a job has failed, it requires, on the average, a lot more local resubmissions than global ones to be successfully executed.

Table 2. Performance of the local strategy when asking to exceed the reliability obtained by the global strategy when $R = 2$

Platform id	Total cases	Percentage of cases requiring more than 2 resubmissions to exceed the reliability of the global strategy	Percentage of cases with better throughput	Overall best throughput ratio
1	87	48.3%	48%	0.5%
2	147	93.9%	35%	4.3%
3	147	91.2%	54%	2.1%
4	147	93.2%	11%	0.7%
5	147	99.3%	16%	1.2%
6	147	100%	22%	0.5%
Overall	822	90.5%	31%	4.3%

6 Conclusion

Failure is an ordinary characteristic of large-scale distributed environments. In this paper we have studied the problem of random brokering on unreliable platforms directly inspired from existing grids such as EGEE. We propose a different approach from the usual trace-based (top-down) where the model is analytically computed from a general model of the environment and the submission strategy.

Here, our bottom-up approach is based on a simple model where incoming jobs are randomly dispatched to computational elements with a probability proportional to the accumulated speed of this element. As we assume that job execution can fail, we study two strategies to improve the reliability (namely local resubmission and global resubmission). For each heuristic we are able to model the saturation ratio (when the incoming load exceed the maximum throughput of the environment), the average waiting time of the jobs and the probability of success of each job. Our experiments show that the proposed models are very realistic. For each of the above metric the models usually predict a very precise value. Furthermore, experiments show that, on the average, the global resubmission is the best strategy as it outperforms, in almost every case, the local resubmission.

Future works are directed towards the evaluation of more metrics such as resource usage, load balance, etc. An other direction of future research is to improve the submission model by adding new laws for inter-arrival or duration or by mixing laws (*i.e.*. some job durations follow an exponential law while others follow a Weibull). The ultimate goal is to be able to come-up with analytical model that would match real traces using this approach.

References

1. Angskun, T., Fagg, G., Bosilca, G., Pjesivac-Grbovic, J., Dongarra, J.: Scalable Fault Tolerant Protocol for Parallel Runtime Environments. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 141–149. Springer, Heidelberg (2006)
2. Berten, V., Goossens, J., Jeannot, E.: On the Distribution of Sequential Jobs in Random Brokering For Heterogeneous Computational Grids. IEEE Transactions on Parallel and Distributed Systems 17(2), 113–124 (2006)
3. Bouteiller, A., Herault, T., Krawezik, G., Lemarinier, P., Cappello, F.: Mpich-v: a multiprotocol fault tolerant mpi. International Journal of High Performance Computing and Applications (2005)
4. Casanova, H., Legrand, A., Quinson, M.: SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In: 10th IEEE International Conference on Computer Modeling and Simulation (March 2008)
5. Costa, G.D., Dikaiakos, M., Orlando, S.: Analyzing the workload of the south-east federation of the egee grid infrastructure. Tech. Rep. TR-0063, Institute on Knowledge and Data Management, CoreGRID - Network of Excellence (February 2007),
`http://www.coregrid.net/mambo/images/stories/TechnicalReports`
`tr-0063.pdf`

6. Enabling Grids for E-sciencE (EGEE), http://www.eu-egee.org/
7. Iosup, A., Dumitrescu, C., Dick, H.J., Epema, H.L., Wolters, L.: How are real grids used? the analysis of four grid traces and its implications. In: GRID 2006, pp. 262–269 (2006)
8. Jensen, H.T., Leth, J.R.: Automatic Job Resubmission in the Nordugrid Middleware. Tech. rep., Aalborg University (2004), http://www.nordugrid.org/documents/jensen_leth.pdf
9. Li, H., Heusdens, R., Muskulus, M., Wolters, L.: Analysis and synthesis of pseudo-periodic job arrivals in grids: A matching pursuit approach. In: CCGRID 2007, pp. 183–196 (2007)
10. Medernach, E.: Workload analysis of a cluster in a grid environment. In: Job scheduling strategies for parallel processing, pp. 36–61 (2005)
11. Rood, B., Lewis, M.J.: Multi-state grid resource availability characterization. In: GRID 2007, pp. 42–49 (2007)
12. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: DSN 2006, pp. 249–258 (2006)
13. TeraGrid, http://www.teragrid.org/

Reliable Parallel Programming Model for Distributed Computing Environments

Jacques M. Bahi, Mourad Hakem, and Kamel Mazouzi

Université Franche Comté, LIFC laboratory
IUT Belfort-Montbéliard, Rue Engel Gros
BP 527 90016 Belfort CEDEX France
{Jacques.Bahi,Mourad.Hakem,Kamel.Mazouzi}@lifc.univ-fcomte.fr

Abstract. With the advent of large-scale heterogeneous platforms such as clusters and grids, resource failures are more likely to occur and have an adverse effect on the applications. Consequently, there is an increasing need for developing techniques to achieve reliability during execution. This paper presents FT-Jace, a new reliable programming model for grid computing environments. FT-JACE achieves reliability in a transparent manner for the programmer. It is based on active replication scheme, capable of supporting r arbitrary fail-silent (a faulty node does not produce any output) and fail-stop (no node recovery) node failures. The strength of our programming environment is that the deployment of the application does not require complicated mechanisms for failure detection. More precisely, node failures are masked and there is no need for detecting and handling such failures. We provide experimental results conducted on Grid'5000[1] platform to demonstrate the usefulness of FT-Jace.

1 Introduction

In large-scale heterogeneous grids, the probability of a failure is much greater than in traditional parallel systems. Consequently, reliability is becoming a crucial area in grid computing since the difficulty for setting up a coherent platform is higher as the number of nodes gets larger. The most popular fault-tolerance mechanism is checkpointing[10,6,7,14,3,8,9], i.e., periodically saving the state of the application on stable storage, usually a hard disk. After a crash, the application is restarted from the last checkpoint rather than from the beginning. Checkpointing is used in grid computing by systems such as Condor [13] , Cactus [1] and MPICH-V2 [6]. One of the disadvantages of checkpointing is its fault tolerance overhead, even if there are no crashes. In addition it assumes a reliable server where checkpoints can be stored. Writing the state of the process to stable storage is the main source of this overhead. Another problem of most checkpointing schemes is the complexity of the crash recovery procedure, especially in dynamic and heterogeneous grid environments where retrieving and transferring the checkpoint data between nodes is non-trivial. The fault tolerant scheme we propose does not suffer from this problem, because crashes can

[1] http://www.grid5000.org

be masked without the need of detecting and handeling them. The likely best-known family of fault tolerance techniques is task replication [5]. There are two main approaches, as described below :

i) Passive replication (primary/backup): This is the traditional fault-tolerant approach where both time and space exclusions are used. The main idea of this technique is that the backup task is activated only if the fault occurs while executing the primary task. This technique assumes that there is a fault detection/recovery mechanism that detects and handles node failures.

ii) Active replication (N-Modular redundancy): This technique is based on space redundancy, i.e., multiple copies of each task are mapped on different nodes, which are run in parallel to tolerate a fixed number of failures. For instance, Genaud and Rattanapoka [11] propose P2P-MPI a Peer-To-Peer framework for robust execution where multiple node failures are considered. Unfortunately, this technique also assumes that there is a fault detection/recovery mechanism that detects and handles node failures. To the best of our knowledge, P2P-MPI is the closest work to the one presented in this paper.

We strongly believe that the active replication scheme is an alternative solution which does not require i) any specific reliable resource to store system states and ii) complicated mechanisms for failure detection/recovery. Our programming environment should provide automatic and transparent mechanisms to improve reliability of the system network.

Our motivation in this work is to propose a mechanism that has smaller overhead and is simpler to implement than more general techniques such as checkpointing. This is the reason why we propose FT-Jace, a solution based on task replication. Such a mechanism can have very low overhead, as no synchronization-coordination between processes is needed and no data needs to be stored on stable storage. The programming model we present in this paper is robust since it resists to eventual failures even without being detected. It could serve as a basis for fault tolerance and self configuration.

The rest of this paper is organized as follows: in Section 2, we outline our fault-tolerance mechanism and describe its implementation. In Section 3, first we give some properties of FT-Jace and next we outline its threads management level and reliability analysis. We present the results of the performance evaluation of our mechanism in Section 4. Finally, we conclude in Section 5.

2 FT-Jace Platform

Our mechanism is an improved version of JACE, a Java programming and executing environment [2] that was introduced/developed to implement efficient asynchronous algorithms as simply as possible. FT-Jace builds a distributed virtual machine, composed of heterogeneous machines scattered over several distant sites. It proposes a simple programming interface to implement applications using the message passing model. The interface completely hides the mechanisms related to asynchronism, especially the communication manager and the global convergence control. In order to propose a more generic environment, FT-Jace also provides primitives to implement synchronous algorithms and a simple mechanism to swap from one mode to another. The improved version that we propose relies on three components: *the daemon, the computing task* and *the spawner*.

2.1 The Daemon

The daemon is the entity responsible for executing user applications. It is a Java process running on each node taking part in the computation. Figure 1 shows the internal architecture of the daemon which is composed of three layers : *the worker service, the application layer* and *the communication layer*.

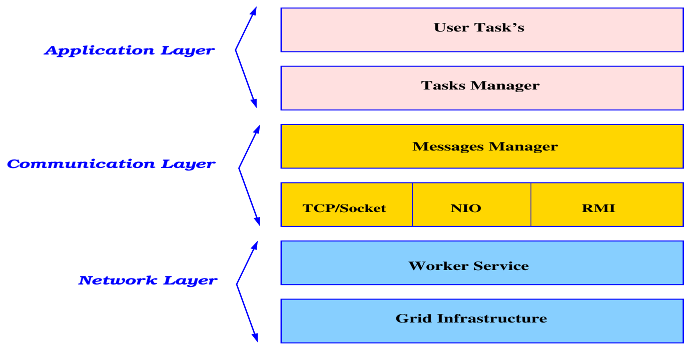


Fig. 1. FT-Jace daemon architecture

The Worker Service. When a daemon is launched, a service is started on it and is continuously waiting for remote connections. This server provides communications between the daemons and the spawner. It is used to manage the FT-Jace environment like for example: initializing the daemons, monitoring and gathering the results.

The Application Layer. This layer provides task-replicas execution and global convergence detection. A daemon may execute multiple task-replicas, allowing to reduce distant communications. It is designed to control the global convergence process in a transparent way. Tasks only compute their local convergence state and call the FT-Jace API to retrieve the global state. The internal mechanism of the convergence detection depends on the execution mode i.e. synchronous or asynchronous.

The Communication Layer. Communications between task-replicas are performed using the message/object passing model. FT-Jace uses waiting queues to store incoming/outgoing messages and two threads (`sender` and `receiver`) to deal with communications. According to the kind of algorithm used, synchronous or asynchronous, queues managements are different. For a synchronous execution, all messages sent by a task-replica must be received by other task-replicas.

Whereas on an asynchronous execution, only the most recent occurrence of a message, with the same source or destination and containing the same type of information is kept, in the queues. The older one, if existing, is deleted.

To not take into account the redundant messages due to replication, each daemon mantains an information table. Each entry is composed of two fields: $< src\_tag >, < counter >$, where *src* is the FT-Jace rank of sender task, *tag* is a tag number of FT-Jace task and *counter* is the number of calling FT_Jace_Send. At the reception of a message, the daemon checks if a message whith the same $< src\_tag >$ field already exists in its information table. If a message exists, it checks the *counter* value. If the counter value of the new message is greater than or equal to the counter value in the table, the message is updated by the newer one. Otherwise, the message is ignored. But if the message does not exists, it simply puts that message in its information table.

For scalability issues and to achieve better performances, the communication layer should use an efficient protocol to exchange data between remote task-replicas. For this reason FT-Jace is based on several protocols : TCP/IP Sockets, NIO (New Input/Output) and RMI. NIO is a Java API (introduced in Java 1.4). It provides new features and improved performances in the areas of buffer management, scalable network and file I/O. The most important distinction between the original I/O library and NIO is how data is packaged and transmitted. Original I/O deals with data in streams, whereas NIO deals with data blocks and consumes a block of data in one step. Furthermore, previously for network applications, users would have had to deal with multiple socket connections by starting a thread for each connection. Inevitably, they would have encountered issues such as operating system limits, deadlocks, or thread safety violations more specially in a large scale context. With NIO, selectors are used to manage multiple simultaneous socket connections on a single thread.

2.2 The Computing Task

As in MPI-like environments, the programmer decomposes the problem to be solved into a set of cooperating sequential tasks. These tasks are executed on the available nodes and invoke special routines to send or receive messages. A `task` is the computing unit in FT-Jace, which is executed like a thread rather than a process. Thus, multiple task-replicas may execute in the same daemon and can share the system resources. The task-replicas running in the same node use shared memory to exchange data.

To write a FT-Jace application, the user simply needs to extend the `Task` class and to define a `run()` method containing its program code. The `Task` class may be considered as the programming interface of FT-Jace. It contains a limited set of methods and attributes dedicated to implement asynchronous/synchronous algorithms in a message passing style. To summarize, we can find:

– the non-blocking send/receive,
– the blocking send/receive (for synchronism),
– the global communications: barrier, broadcast, rendezvous,

– the convergence control,
– the finalization.

2.3 The Spawner

The spawner is the entity that effectively starts the user application. After starting daemons on all nodes, computations begin by launching the spawner program with the following parameters:

– the number of task-replicas to be executed;
– the URL of the task byte-code;
– the parameters of the application;
– the list of target daemons;
– the mapping algorithm (round robin).

Then, the spawner broadcasts this information to all the daemons. Now, when a task-replica is spawned, an identification number (task ID or rank) is assigned to it. This number is an integer whose value ranges from 0 to $n(r+1)-1$, with n being the global number of tasks in the FT-Jace application $\mathcal{A}$. This mapping is done by FT-Jace and by default uses a round robin algorithm (see Figure 2).

```
$n \leftarrow$ number of FT-Jace tasks;
$r \leftarrow$ fault tolerance degree;
$m \leftarrow$ number of available nodes;
$k \leftarrow 0$;
for $i = 0$ to $n$ do
  for $j = 0$ to $r$ do
    rank($t$) $= i + n \times j$;
    map a task $t$ on a node $k$;
    $k = (k+1) \bmod m$;
  end for
end for
```

Fig. 2. Rank assignement algorithm

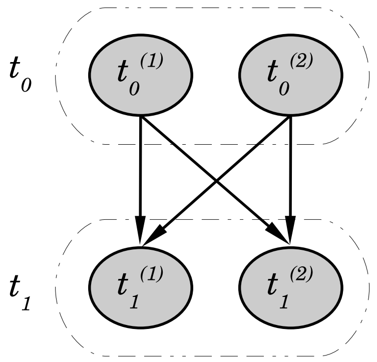


Fig. 3. A message sent from task t_0 to t_1

Task replication. Our implementation is based on an active replication scheme, capable of supporting r arbitrary fail-silent/fail-stop node failures. This replication management is absolutely transparent for the programmer. When specifying a desired number of tasks, the programmer requests the system to run for each task an arbitrary number of copies called replicas. In the following, we denote by $\mathcal{S}(t)$ the set of $r+1$ replicas of tasks t. Also, we denote by $t^{(k)}$ those replicas, for $1 \le k \le r+1$. Thus, $\mathcal{S}(t) = \{t^{(1)}, t^{(2)}, \ldots, t^{(r+1)}\}$. $P\left(t^{(k)}\right)$ is the node on which replica $t^{(k)}$ is assigned by the spawner. The task-replicas are run in parallel on different nodes (space exclusion, see Property 1) since the goal is to allow the continuation of the execution even if some node failures occur.

Now, we explain how the task replication is related to FT-Jace rank assignment. Our model uses an active replication scheme to tolerate a fixed number r of failures. The spawner takes the number of FT-Jace tasks n and the fault tolerance degree (or reliability factor) r as arguments and then tries to mappe the task-replicas using a round-robin matching as shown in Figure 2. Since all replicas of the receiver side receive the sent message from all replicas of the sender side, there is no need to manage/coordinate the replicas behavior. The communication scheme is kept coherent with the semantics of the original fault free version of FT-Jace application.

In Fig 3, we show the FT-Jace rank matching computed by the spawner when the programmer executes an FT-Jace application requiring two tasks and fault tolerance degree of 1, i.e., the equivalent of **FT-Jacerun -np 2 -r 1**. Rank 0 refers to the task which collects the results. It is not executed at requesting user's node. Thus, it is also replicated. Therefore, the number of task-replicas involved for execution is $n(r+1)$.

3 Analysis of the FT-Jace Environment

In this section, first we present some properties of our fault tolerant scheme, and express its communication/computation overhead, next we detail its threads management level and reliability analysis.

Property 1. *For an active replication scheme, a task t_i in the FT-Jace application $\mathcal{A}$ is guaranteed to execute in the presence of r permanent faults if and only if $P(t_i^k) \neq P(t_i^{k'})$, for $1 \leq k, k' \leq r+1$.*

Proof. If r nodes fail, then there is $P(t_i^z), 1 \leq z \leq r+1$ which did not fail, and therefore $P(t_i^z)$ executes successfully since there are $r+1$ copies of t_i assigned to $r+1$ different nodes. However, if there is a node $P(t_i^k), 1 \leq k \leq r+1$, such that $P(t_i^k) = P(t_i^z) = P^*$ and P^* fails, then neither t_i^k nor t_i^z can execute successfully.

Property 2. *Let t and t_* two tasks involved in communication. If a replica of task t and a replica t_*^z of t_* are mapped on the same node P, then there is no need for other replicas of t_* to send data to node P.*

Proof. if P is operational, then the replica of t on P will receive the message from t_*^z (intra-node communication). Otherwise, P is down and does not need to receive anything.

Proposition 1. *If at most r node failures occur in the system network, then the matching computed by the spawner remains valid and resists to r failures.*

Proof. FT-Jace is based on an active replication scheme with space exclusion. Thus according to Property 1, each task is replicated $r+1$ times onto $r+1$ distinct nodes. We have at most r node failures at the same time. So at least one copy of each task is executed on a fault free node.

Latency bounds: Let $\hat{t}$ be the last task to be executed in the application. The lower bound $\mathcal{M}^*$ of the response time (latency) of the application can be achieved if no node permanently fails during the execution of the application. It is defined as follows:

$$\mathcal{M}^* = \max_{\hat{t}_i \in \mathcal{A}} \left\{ \min_{1 \leq k \leq r+1} \{\mathcal{F}(\hat{t}_i^k, P(\hat{t}_i^k))\} \right\} \tag{1}$$

To compute the upper bound of the response time $\mathcal{M}$, which is achieved in the presence of r permanent failures, we use the following formula:

$$\mathcal{M} = \max_{\hat{t}_i \in \mathcal{A}} \left\{ \max_{1 \leq k \leq r+1} \{\mathcal{F}(\hat{t}_i^k, P(\hat{t}_i^k))\} \right\} \tag{2}$$

Proposition 2. *The latency achieved by FT-Jace is $\mathcal{L} \leq \mathcal{M}$ despite r permanent failures.*

Proof. Since the application $\mathcal{A}$ can be represented as a Directed Asyclic Graph (DAG), the proof is similar to that presented in [5].

– **Communication/computation overhead:** To resist to r failures, each task $t \in \mathcal{A}$ is replicated $r+1$ times. Allocating many replicas of each task will increase the total number of communications required by the application: we move from m communications in a matching with no replication (fault free application), to $m(r+1)^2$ with replication (fault tolerant application), a quadratic increase. But in fact our mechanism exploits threads to manage both communications and computations (see below). This leads to a significant reduction of fault tolerance overhead.

– **Threads management level:** The initial version of Jace is a multi-threaded environment which runs three threads in parallel: the computation thread, the sender thread and the receiver thread. The new version that we prpose redefines the threads management policy, in partiular when several task-replicas runs on the same node. This case becomes more and more relevant due to the generalization of multi-cores processors. So, to avoid synchronizations between the computation and the communication threads a unique sending queue is used. So, when k task-replicas running on a node we get 1 sending queue and k receiving queues. The threads scheduling is also modified. Indeed, according to several experimentations it appears that the *Asynchronous Iterations Asynchronous Communications* (AIAC) algorithms are more efficient when the sender thread is of high priority. This can be explained by the fact that in this case, messages are more frequently updated and consequently the algorithm converges more quickly.

– **Improving reliability:** For serial and parallel configurations of components which turn up in many different systems, the reliability of each component can be given in a variety of ways: mean time to failure $MTTF$, reliability percentage over a period of time, failure rate, etc. For complex systems, a component's failure rate is measured meticulously using extended benchmarks and simulations. Modern fail-silent processors can have a failure rate of the order of 10^{-6}

per hour [12]. The overall reliability of a system which is composed of a chain of components can be calculated as the probability of all components executing successfully. In other words, it is the product of the individual reliabilities: $\mathcal{R}_{syst} = \prod_i \mathcal{R}_i$. For a system of components running in parallel, the overall reliability is 1 minus the probability that all components fail:

$$\mathcal{R}_{syst} = 1 - F_{syst} = 1 - \prod_i F_i = 1 - \prod_i \left(1 - \mathcal{R}_i\right)$$

Node failures in the system network are assumed to be statistically independent and follow an exponential law with a probability density function (PDF) $f(t) = \lambda e^{-\lambda t}, t \geq 0$, where λ is the constant failure rate. Thus, from a node's failure rate, we can derive the reliability of that node over a period of time t as follows:

$$\mathcal{R}(T > t) = 1 - P(T \leq t) = 1 - \int_0^t f(t) = e^{-\lambda t}$$

where T is a random variable associated to a node. Thus, if a node is estimated to have a failure rate of 5 failures every 10000 hours, then its reliability over a 24 hour period is calculated to be approximately 99%.

For a mapping without repliaction, the execution of the application is successful if and only if each node is operational (no failure) while it is executing its assigned tasks. So, the reliability of the system network is the probability that the application $\mathcal{A}$ can run successfuly during the mission. Formally: $\mathcal{R}_{\mathcal{A}} = \prod_{i=1}^{n} \mathcal{R}_i$.

It follows that for a mapping with replication, successful execution of the application $\mathcal{A}$ requires that at least one replica of each task $t_i \in \mathcal{A}$ be executed successfuly: $\mathcal{R}_{\mathcal{A}} = \prod_i R_i = \prod_i \left(1 - \prod_j \left(1 - \mathcal{R}_j\right)\right), 1 \leq i \leq n, 1 \leq j \leq r + 1$.

4 Experimental Results

In this section, we report the experiments we have performed on the French Grid'5000[2] platform. We have used 50 nodes located at Sophia-Antipolis. The nodes used are equipped by bi-core AMD Opteron 2.0 GHZ and connected in Gigabit Ethernet. We have implemented both synchronous and asynchronous iterative mode of Jacobi method to solve linear systems ($Ax = b$). For more details concerning this method, readers are invited to consult [4]. The size of the matrix A is $10^8 \times 10^8$.

As a first evaluation, the metrics which caracterize the performance of FT-Jace are the achieved execution time (latency) and the overhead due to the active replication scheme. The overhead is computed as $\text{Overhead}_{FT-Jace} = \frac{\mathcal{L}_{FT-Jace} - \mathcal{L}_{FF}}{\mathcal{L}_{FF}}$, where $\mathcal{L}_{FT-Jace}$ is the execution time achieved by FT-Jace, and $\mathcal{L}_{FF}$ the execution time of the fault free mapping defined as the matching generated by the spawner without replication, assuming that the system is completely safe, setting $r = 0$. We let $r = \{1, 2, \ldots, 10\}$, which corresponds to a reasonable number of failures for an architecture of 50 nodes.

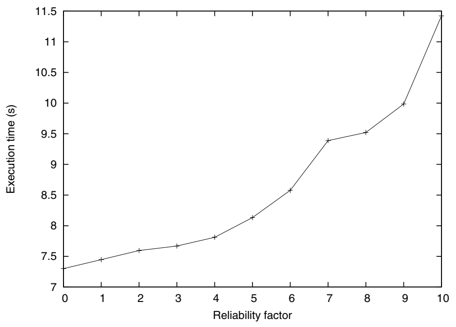


Fig. 4. Asynchronous mode

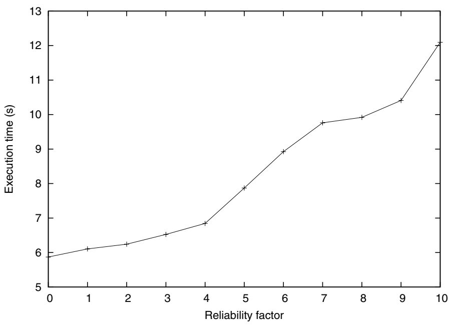


Fig. 5. Synchronous mode

Table 1. Fault tolerance overhead (%)

Reliability factor (r)	2	4	6	8	10
Asynchronous mode	0.04	0.06	0.17	0.30	0.56
Synchronous mode	0.06	0.16	0.52	0.69	1.06

As expected, we observe from figures 4, 5 and table 1 that we deal with two conflicting objectives. Indeed, the fault tolerance overhead increases together with the number of supported failures. We also see that latency increases together with the presecribed fault tolerance degree. However, FT-Jace achieves a really good performance, which is very close to the fault free version. In addition, the fault tolerance overhead induced by the active replication scheme is reasonable and increases slightly as the reliability factor (fault tolerance degree) goes up.

We did not have time to test the impact of crashes on the preformance of FT-Jace, but we expect as shown in [5] that when the number of failures increases, there will not be really much difference in the increase of the latency achieved by FT-Jace, compared to the latency achieved with 0 crash (the lower bound). This is explained by the fact that the increase of latency is already absorbed by the replication done previously, in order to resist to eventual failures.

We are currently working on: (i) evaluating the impact of crashes on the achieved latency and (ii) testing FT-Jace in a large scale context with more than 200 nodes and with some other scientific intensive applications. These tests will allow us to better evaluate/analyse the behaviour of FT-Jace.

5 Conclusion

In this paper, we have presented a new parallel programming model for grid environments. It is based on an active repliaction scheme and does not require complicated mechanism for fault detection/recovery. Our fault-tolerant scheme has the potential to become a viable platform for Grid applications as no synchronisation between tasks is needed and no data needs to be stored on stable

[2] http://www.grid5000.org

storage. In addition, it exploits threads to manage both communications and computations, this leads to a significant reduction of fault tolerance overhead.

An extension of FT-Jace would be to change the mapping algorithm of the spawner by an effective scheduling strategy that take into account both algorithmic and architectural characteristics (resources capability and reliability) to achieve a good mapping of task-replicas to nodes. We would then need to solve a challenging bi-criteria optimization problem (latency and reliability).

References

1. Allen, G., Benger, W., Goodale, T., Hege, H.-C., Lanfermann, G., Merzky, A., Radke, T., Seidel, E., Shalf, J.: The cactus code: A problem solving environment for the grid. In: HPDC, pp. 253–260 (2000)
2. Bahi, J.M., Couturier, R., Laiymani, D., Mazouzi, K.: Java and asynchronous iterative applications: large scale experiments. In: IPDPS, pp. 1–7 (2007)
3. Bahi, J.M., Couturier, R., Vuillemin, P.: JACEV: A programming and execution environment for asynchronous iterative computations on volatile nodes. In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006. LNCS, vol. 4395, pp. 79–92. Springer, Heidelberg (2007)
4. Bahi, J.M., Vivier, S.C., Couturier, R.: Parallel Iterative Algorithms: from sequential to grid computing. Numerical Analysis & Scientific Computating, vol. 1. Chapman & Hall/CRC, Boca Raton (2007)
5. Benoit, A., Hakem, M., Robert, Y.: Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems. J. Parallel Comput. 35(2), 83–108 (2009)
6. Bouteiller, A., Cappello, F., Hérault, T., Krawezik, G., Lemarinier, P., Magniette, F.: MPICH-V2: a fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. In: Supercomputing (2003)
7. Bouteiller, A., Desprez, F.: Fault tolerance management for a hierarchical GridRPC middleware. In: CCGRID, pp. 484–491 (2008)
8. Buntinas, D., Coti, C., Hérault, T., Lemarinier, P., Pilard, L., Rezmerita, A., Rodriguez, E., Cappello, F.: Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI protocols. Future Generation Comp. Syst. 24(1), 73–84 (2008)
9. Charr, J.-C., Couturier, R., Laiymani, D.: JACEP2P-V2: a fully decentralized and fault tolerant environment for executing parallel iterative asynchronous applications on volatile distributed architectures. In: GPC (2009)
10. Fagg, G.E., Dongarra, J.: FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In: Dongarra, J., Kacsuk, P., Podhorszki, N. (eds.) PVM/MPI 2000. LNCS, vol. 1908, pp. 346–353. Springer, Heidelberg (2000)
11. Genaud, S., Rattanapoka, C.: Fault management in P2P-MPI. In: Cérin, C., Li, K.-C. (eds.) GPC 2007. LNCS, vol. 4459, pp. 64–77. Springer, Heidelberg (2007)
12. Girault, A., Saule, E., Trystram, D.: Reliability versus performance for critical applications. JPDC 69(3), 326–336 (2009)
13. Litzkow, M.J., Livny, M., Mutka, M.W.: Condor - a hunter of idle workstations. In: ICDCS, pp. 104–111 (1988)
14. Louca, S., Neophytou, N., Lachanas, A., Evripidou, P.: MPI-FT: Portable fault tolerance scheme for MPI. Parallel Processing Letters 10(4), 371–382 (2000)

Second Workshop on Productivity and Performance (PROPER 2009)

PROPER 2009: Workshop on Productivity and Performance – Tools for HPC Application Development

The PROPER workshop addresses the need for productivity and performance in high performance computing. Productivity is an important objective during the development phase of HPC applications and their later production phase. Paying attention to the performance is important to achieve efficient usage of HPC machines. At the same time it is needed for scalability, which is crucial in two ways: Firstly, to use higher degrees of parallelism to reduce the wall clock time, i.e. the response time for the user. And secondly, to cope with the next bigger problem, which requires more CPUs, memory, etc. to be able to compute it at all.

Support for the user via specialized tools is essential for productivity and performance. Therefore, the workshop covers tools and tool approaches for parallel program development and analysis, for debugging and correctness checking, and for performance measurement and evaluation. Furthermore, it provides an opportunity to report successful optimization strategies with respect to scalability and performance.

All of this year's successful contributions focus on performance analysis tools and new ideas for their design. They can be divided into the two traditional and fundamental methods, profile accumulation and event-trace recording. Yet, all contributions indicate a trend of convergence between both methods. On the one hand, sophisticated reduction of event traces allows higher scalability while keeping the high level of detail event tracing is known for. On the other hand, enhancement of the profiling concept allows to provide more fine grained performance data or specific information that is not delivered by traditional profiling tools. This increases the level of detail while retaining the inherently good scalability. Even though profiling and tracing are generally seen as complementary methods, the paper by Fürlinger et. al. goes so far as to attempt a synthesis of both.

We would like to thank all the authors for their very interesting contributions and their presentations during the workshop. And furthermore, we would like to thank the EuroPar 2009 organizers for their support and for the chance to offer the PROPER workshop in conjunction with this attractive conference.

Andreas Knüpfer
Jens Doleschal
Matthias Müller
Felix Wolf

Argument Controlled Profiling

Tilman Küstner, Josef Weidendorfer, and Tobias Weinzierl

Technische Universität München, Germany
{kuestner,weidendo,weinzier}@in.tum.de

Abstract. Profiling tools relate measurements to code context such as function names in order to guide code optimization. For a more detailed analysis, call path or phase-based profiling enhances the context by call chains or user defined phase names, respectively. In this paper, we propose *argument controlled profiling* as a new type of context extension using the value of function arguments as part of the context. For a showcase simulation code, we demonstrate that this simplifies and enriches the understanding and analysis of code—in particular recursive functions. Due to the new profiling technique, we found optimizations resulting in more than 16% runtime improvement. Argument controlled profiling is implemented as extension of Callgrind, a simulation-based profiling tool using runtime instrumentation.

1 Introduction

As program codes today typically consist of millions of lines of code, as computer architectures are diverse and sophisticated, and as there is a high pressure on the developer to deliver the whole software on time, writing fast code out of the box is a complex, demanding, almost impossible challenge. On the one hand, it is hard to predict in which code parts most of the runtime is spent. On the other hand, these code parts have to be tailored to the actual hardware. Profiling is the tool of choice to identify the code parts and to get hints on how to do an optimization.

However, use of direct or indirect recursion can make the results of a profiler difficult to understand. To provide useful data such as inclusive cost, functions which are part of recursive cycles need to be suppressed by introducing artificial cycle functions comprising all functions of this recursive cycle. They are identified by strongly connected components in the dynamic call graph [6]. Unfortunately, this discards a lot of information and, sometimes, renders results useless for analysis. The solution is to relate event counts measured by the profiling tool not only to the function name, but to a more elaborated context which better describes the code position. A similar argument holds for non-recursive functions whose runtime behavior depends significantly on their arguments.

One typical extension is the *calling context* [1, 15] which takes the call chain to the current function into account. Unfortunately, this does not help when the behavior of a recursive function does not depend on the call chain leading to it, but on values in data structures instead. This is often the case, and conceals important information. For example, in a code where a recursive function is visiting nodes of a tree or graph structure in some order, it is valuable to to see how often one node type follows after another node type, as this helps optimizing often executed code paths.

Thus, it is useful to enhance the function context by incorporating a program state. To distinguish a function with respect to a state, it has to be specified by the user before entering the function. A natural choice is to use the value of function arguments. In the output of the profiling tool, the same function with different states then is treated as individual symbols. Thus, cost is also assigned to different symbols. This is the main idea presented in this paper: the *argument controlled profiling*.

We apply this technique to our cache simulation tool Callgrind [13], which uses the runtime instrumentation framework Valgrind [9]. The simulation approach using runtime instrumentation has two specific advantages: first, any overhead generated by collecting separate metrics for an increased number of symbols does not matter, as runtime overhead does not influence results of the simulator. Second, it is important to selectively use this feature as the number of symbols can explode. The runtime instrumentation approach allows to do multiple profiling runs with different selective configurations using the same binary without recompilation. The latter is important if long recompilation times required for different instrumentation would be prohibitive for an in-depth analysis.

The remainder is organized as follows: First, we discuss related work. Second, the usage of argument controlled profiling is described with a small example. Third, we provide details on our prototype implementation. A practical showcase is given afterwards, together with the applied optimization approach and results on runtime improvements. A short conclusion with an outlook closes the discussion.

2 Related Work

The simplest useful result of a profiling tool is the *flat profile*. All performance cost such as time (clock ticks) or cache events is related to code which triggered the cost. However, entries in a flat profile may point to code parts which reside in a library unavailable for modifications, or they carry no potential for optimization in themselves. Here, tracking back the call stack for optimizable code parts, i.e. lines which invoke the expensive functions, is valuable. For this, the context is enriched by the call path: it becomes a calling context. A lot of papers in the area of profiling tools discuss the efficient collection of full call paths (starting at the entry point of the program), using precise yet low-overhead measurement strategies [1, 11, 5, 15, 3]. Many profiling tools provide the calling context on request [7, 10, 13, 12, 4].

For tools performing real-time measurements, there is always a tradeoff between accuracy and completeness because of the influence of measurement overhead. A good compromise is to allow selective instrumentation. The developer helps minimizing overhead by providing hints on what type of information she is looking for before the measurement starts. The TAU Performance System [10] implements so-called *phase-based profiling* [8]. The developer can indicate logical phases of her program by calling a profiling interface (API). These phase names, arbitrary strings, can be constructed to contain function arguments. In contrast to our approach, this technique requires the user to manually modify the source code and recompile it. Another selective instrumentation method is *incremental call path profiling* [2], where specified functions are dynamically instrumented to do full stack walks for determining the calling context.

Sampling is a strategy not based on instrumentation. It allows the overhead to be tunable, but provides statistical results. Only every n-th occurrence of an event of interest is collected. To get the calling context at sample points, stack walks at arbitrary points of execution need to be supported. Some tools such as Intel's PTU [4] rely on debug information generated by the compiler. However, not all compilers produce reliable debug information. By contrast, hpcrun [12] uses binary analysis to generate meta information needed for stack walks.

A lot of the tools mentioned above preserve recursive invocations in the calling context. However, if it is known that the behavior of a recursive function does not depend on recursion depth, an overwhelming amount of unneeded information is generated.

3 Usage and Example

Argument controlled contexts allow to distinguish profiling results of the same function according to function arguments. Yet, this can not be done for every argument-value combination of every function, as this would lead to an explosion of profiling results. Instead, the programmer needs to specify that she wants a distinction for a given function according to selected values of arguments.

In our prototype extension of Callgrind, we support the command line option `--separate-par=`*function*`:`*num*[`:`b_1[`,`b_2...]]. Here, $function$ is the name of the function for which we request the creation of an enhanced context: the *num*-th 4-byte-value on the stack, interpreted as 4-byte integer value (which also works for booleans), is to be incorporated into the context name. Optionally, b_i values can be given, representing bucket borders. E.g. for value x and borders $(5, 10)$, the ranges $x <= 5$, $5 < x <= 10$, and $10 < x$ are distinguished.

We consider the following 2-dimensional Fibonacci-like function

```
int fib2(int i, int j) {
  if ((i<2) || (j<2)) return 1;
  return fib2(i, j-1) + fib2(i-1, j) + fib2(i-1, j-1);
}
```

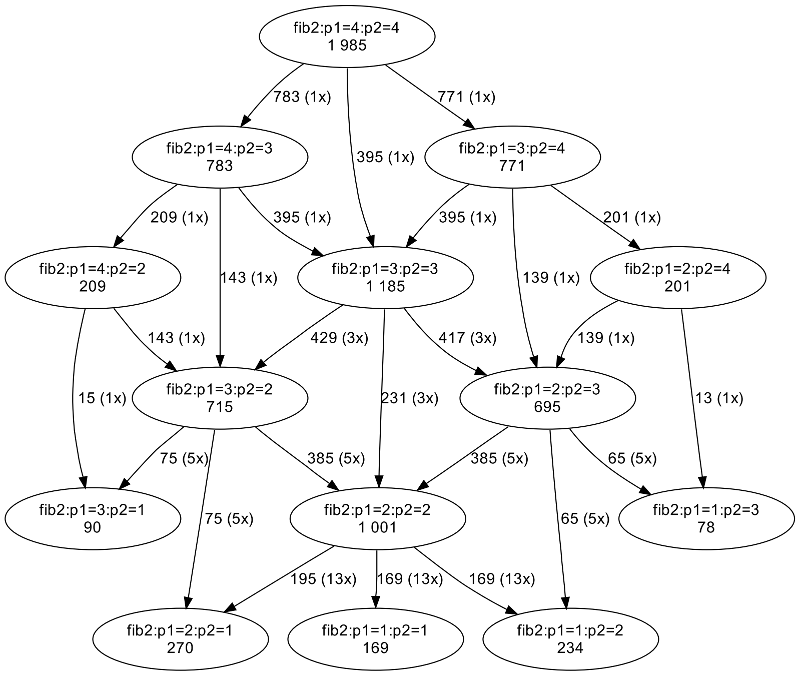


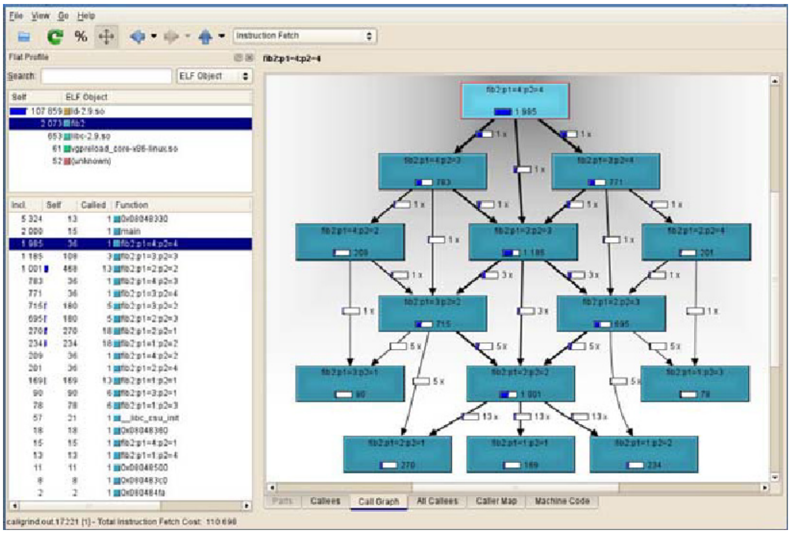
Fig. 1. The call graph of `fib2(4,4)`

together with the `main` function calling `fib2(4,4)`. Executing

```
> valgrind --tool=callgrind \
         --separate-par=fib2:1 --separate-par=fib2:2 ./fib2
```

produces among others profiling data visualized in Fig. 1, if we select the symbol "`fib2:p1=4:p2=4`" to be displayed. Each node represents a profile for function `fib2`, but is distinguished by the requested argument values. Edges between nodes represent calls with the call count shown in parenthesis. The number given in the nodes and next to edges is the inclusive cost for the event "Instructions Executed". This is the event collected by Callgrind when cache simulation is switched off.

Fig. 1 is generated by the "Export Graph" functionality of our profile visualization tool KCachegrind [13]. The screenshot in Fig. 2 shows on the left a list of functions (with `fib2:p1=4:p2=4` selected). The functions are grouped according to the "ELF object" they reside in. These groups are shown in the list above the functions. The selected group "fib2" is the main executable, i.e. the function list only shows functions residing in the binary "fib2". On the right, the interactive call graph visualization for the selected function is displayed. This is just one of many visualization types supported by KCachegrind, such as caller/callee lists, *tree map* visualizations of the callee tree, or source/assembly annotations.



Self	ELF Object
107 859	ld-2.9.so
2 073	fib2
653	libc-2.9.so
61	vgpreload_core-x86-linux.so
52	(unknown)

Incl.	Self	Called	Function
5 324	13	1	0x08048330
2 009	15	1	main
1 985	36	1	fib2:p1=4:p2=4
1 185	109	3	fib2:p1=3:p2=3
1 001	468	13	fib2:p1=2:p2=2
783	36	1	fib2:p1=4:p2=3
771	36	1	fib2:p1=3:p2=4
715	180	5	fib2:p1=3:p2=2
695	180	5	fib2:p1=2:p2=3
270	270	18	fib2:p1=2:p2=1
234	234	18	fib2:p1=1:p2=2
209	36	1	fib2:p1=4:p2=2
201	36	1	fib2:p1=2:p2=4
169	169	13	fib2:p1=1:p2=1
90	90	6	fib2:p1=3:p2=1
78	78	6	fib2:p1=1:p2=3
57	21	1	__libc_csu_init
18	18	1	0x08048360
15	15	1	fib2:p1=4:p2=1
13	13	1	fib2:p1=1:p2=4
11	11	1	0x08048500
8	8	1	0x080483c0
2	2	1	0x080484fa

callgrind.out.17221 [1] - Total Instruction Fetch Cost: 110 698

Fig. 2. Screenshot of KCachegrind with call graph view

4 Implementation

For Callgrind, it is important to be able to relate the effect of every memory access to a context (e.g. function name, calling context, thread number, and so forth). Thus, it is helpful to have the context readily available whenever an event is triggered. The calling context is updated every time a function is entered or left. This is done by code inserted by runtime instrumentation. The first time a function (more precisely, a *basic block*) is executed, Valgrind forwards the code to Callgrind which inserts instrumentation as needed. The instrumented code then is put into a *translation cache* and executed instead of the original. At instrumentation time (i.e. only once per *basic block*), the function name is related to the code by a lookup in the debug information. At execution time, inserted code can easily detect when the currently active function changes. Whether this change was triggered by a call or return event is heuristically determined with the help of the instruction opcode, the value of the stack pointer, and a shadow call stack data structure. Whenever the tool detects that a new function is entered or left, the above mentioned calling context is updated.

We modified the existing implementation in the following way to allow for argument controlled profiling. On entering a function selected by `--separate-par`, the inserted code does not directly push the symbol name of the entered function to the calling context, but generates a new symbol by appending argument value information to the function name. At this point in time, the value of the stack pointer is readily available to allow access to the function arguments. Finally, the new symbol name is used to update the calling context.

5 Case Study

In the following, we optimize a PDE solver working on adaptive Cartesian grids [14], and we place special emphasis on the benefit of the argument controlled profiling throughout the manual optimization efforts. Although this is an example from scientific computing only, similar rationale, reasonings, and techniques apply to many fields of application.

The solver traverses a tree-like data structure (Fig. 3) with a recursive algorithm which is split up among several routines, and we pick out and concentrate on one particular subroutine. This routine is computationally demanding, as it realizes both the handling of the adaptivity as well as structure of the grid—the grid structure is analyzed, data is preprocessed, and the recursive calls are invoked—and the grid changes due to refinements as well as coarsenings. The tree data structure may change throughout the computation and the routine has to implement the corresponding data transitions and modifications.

The analyzed code part accepts two parameters indicating, on the one hand, whether the global tree will change at all—an information available due to the data structure's public signature—and, on the other hand, whether the current recursion step corresponds to a leaf, i.e. whether it invokes further recursive function calls or not. We run Callgrind, make it distinguish the values of both boolean arguments, and end up with four different parameter constellations for the observed function. The profiling reveals that the first boolean argument holds for most of the recursive calls. Furthermore, the control flow of a recursion tail identified by the latter argument differs significantly from the control flow of all other recursion steps. Entry conditions for some basic blocks never hold. Consequently, these paths are never executed, and the operations to evaluate the entry conditions—they are realized as calls to small helper routines—are obsolete.

A simple refactoring due to this insight delivered by the new tool facility provides two variants of the original code: A standard variant and a variant coming into play whenever the tree data structure does not change (according to the first function argument) or whenever the end of the recursion is reached. We hereafter continue to analyze the effect of the recursion state/flag combination on the individual traversal code parts and apply similar optimizations on

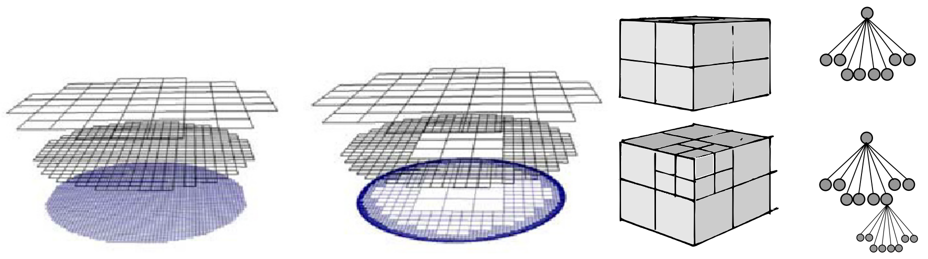


Fig. 3. Cascade of regular 2D Cartesian grids (left), adaptive Cartesian grids (middle), and 3D grids with corresponding tree data structure (right)

several code fragments, i.e. we remove, for one code variant, unnecessary calls to helper functions wherever possible. While all these optimized variants can be derived manually, Callgrind's argument controlled profiling allows us to quantify the performance improvement a priori and often identifies code parts we had not thought about before. The quantification is particularly important if the optimization leads to code duplication, i.e. if it worsens the code's quality. Here, it is important to balance two contradicting coding principles: software quality and performance. Finally, spending only three development days on that kind of optimizations reduced the code's runtime by a factor of more than 16%.

6 Conclusion and Future Work

Argument controlled profiling is a useful feature for analyzing code which makes heavy use of recursive functions. Such code structures are e.g. found within sophisticated PDE solvers where the use of recursive data structures is mandatory for the implementation of adaptive grids. In an industry cooperation, we currently analyze a code which is another candidate for argument controlled profiling: Here, expressions represented by abstract syntax trees (AST) are evaluated. Our approach is valuable in finding often used node type sequences which are candidates for new *super node* types, potentially shrinking the AST—and thus, evaluation time—significantly. To sum it up, we believe that our new profiling technique is of great value for any application traversing tree or graph structures, but also generally for functions whose runtime behavior depends on its arguments.

The current Callgrind prototype is available at `http://mmi.cs.tum.edu`. However, to include our extension into future Callgrind releases, we are going to make it user friendly, flexible, and portable. The user wants to specify function arguments by name. Further, the tool should support more data types (it currently solely works with integer types). Finally, it should run on all architectures supported by Valgrind (currently, it is Linux/x86 only).

Profiling using architecture simulation can be cumbersome to apply to parallel codes. However, for sequential optimization of parallel code, it is usually possible to profile the one-process special case, where our prototype works fine. For future work, it would be interesting to implement our technique also for profilers using real-time measurement, e.g. sampling tools on Linux (such as OProfile [7]). This would make argument controlled profiling applicable also for usage with parallel application runs. The key here is to efficiently and reliably check whether a given instruction address (in the IP register or a return address from the stack) is part of a function whose symbol should be augmented by the value of a function argument, and how to get at this value.

Acknowledgment

This work was supported by the International Graduate School of Science and Engineering (IGSSE), a Ph.D. program at Technische Universität München. This

program emphasizes the cooperation of interdisciplinary research teams. In such a cooperation, the difficulty of analyzing a scientific code—the one used as showcase in this paper—came up and motivated the development of the presented profiling technique.

References

1. Ammons, G., Ball, T., Larus, J.R.: Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling. In: Proceedings of PLDI '97 (June 1997)
2. Bernat, A.R., Miller, B.P.: Incremental call-path profiling. Concurrency and Computation: Practice and Experience 19, 1533–1547 (2007)
3. Bond, M.D., McKinley, K.S.: Probabilistic calling context. In: Proceedings of OOPSLA'07, Montreal, Quebec, Canada (October 2007)
4. Intel Corporation. Intel Performance Tuning Utility, `http://software.intel.com/en-us/articles/intel-performance-tuning-utility`
5. Froyd, N., Mellor-Crummey, J., Fowler, R.: Low-overhead call path profiling of unmodified, optimized code. In: Proceedings of ICS'05, Cambridge, MA (June 2005)
6. Graham, S., Kessler, P., McKusick, M.: GProf: A call graph execution profiler. In: SIGPLAN: Symposium on Compiler Construction, pp. 120–126 (1982)
7. Levon, J.: OProfile, a system-wide profiler for Linux systems, `http://oprofile.sourceforge.net`
8. Malony, A.D., Shende, S.S., Morris, A.: Phase-based parallel performance profiling. In: Proceedings of ParCo'05, Jülich, Germany. Parallel Computing: Current & Future Issues of High-End Computing, vol. 33, pp. 203–210 (2006)
9. Nethercote, N., Seward, J.: Valgrind: A framework for heavyweight dynamic binary instrumentation. In: ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007), San Diego, California, USA (June 2007)
10. Shende, S., Malony, A.D.: TAU: The TAU parallel performance system. International Journal of High Performance Computing Applications 20(2), 287–311 (2006)
11. Spivey, J.M.: Fast, accurate call graph profiling. Software – Practice Experience 34, 249–264 (2004)
12. Tallent, N.R., Mellor-Crummey, J.M., Fagan, M.W.: Binary analysis for measurement and attribution of program performance. In: Proceedings of PLDI'09, Dublin, Ireland (June 2009)
13. Weidendorfer, J., Kowarschik, M., Trinitis, C.: A tool suite for simulation based analysis of memory access behavior. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3038, pp. 440–447. Springer, Heidelberg (2004)
14. Weinzierl, T.: A Framework for Parallel PDE Solvers on Multiscale Adaptive Cartesian Grids. Dissertation, Institut für Informatik, Technische Universität München, München (2009)
15. Zhuang, X., Serrano, M.J., Cain, H.W., Choi, J.-D.: Accurate, efficient, and adaptive calling context profiling. In: Proceedings of PLDI'06, Ottawa, Ontario, Canada, June 2006. ACM, New York (2006)

Detailed Performance Analysis Using Coarse Grain Sampling

Harald Servat, Germán Llort, Judit Giménez, and Jesús Labarta

Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
c/Jordi Girona, 31
08034 - Barcelona, Catalunya, Spain
{harald.servat,german.llort,judit.gimenez,jesus.labarta}@bsc.es

Abstract. Performance evaluation tools enable analysts to shed light on how applications behave both from a general point of view and at concrete execution points, but cannot provide detailed information beyond the monitored regions of code.

Having the ability to determine when and which data has to be collected is crucial for a successful analysis. This is particularly true for trace-based tools, which can easily incur either unmanageable large traces or information shortage.

In order to mitigate the well-known resolution *vs.* usability trade-off, we present a procedure that obtains fine grain performance information using coarse grain sampling, projecting performance metrics scattered all over the execution into thoroughly detailed representative areas.

This mechanism has been incorporated into the MPItrace tracing suite, greatly extending the amount of performance information gathered from statically instrumented points with further periodic samples collected beyond them.

We have applied this solution to the analysis of two applications to introduce a novel performance analysis methodology based on the combination of instrumentation and sampling techniques.

1 Introduction

Performance evaluation tools are able to characterize the behavior of an application using different mechanisms. However, the details obtained by these tools are strictly limited to the monitored regions of code. In order to identify the precise behavior of the application, it is required to add more and more monitors, with the consequent overhead and larger volume of data generated.

Instrumentation and sampling are two different mechanisms used to monitor applications. Instrumentation allows injection of monitors in specific points of the target application simply modifying the source code, compiling the application with special flags or specific libraries, or by using instrumentation packages like DynInst [7]. On the other hand, sampling can be used to trigger monitors at intervals or by external events and it is not related to any specific application point.

We are confident that performance analysis using traces is the best approach to detail time and space variations of the behavior of applications, which can be easily masked out by profilers while summarizing information. In addition, trace visualizers provide further qualitative and quantitative analysis [17,15].

Most performance tools based on traces [22,19,1,2] rely on instrumentation to detail performance characteristics of the applications. Such tools cannot provide information beyond the instrumented points, and the granularity and size of the resulting trace strictly depends on the application structure.

Our main objective is to use basic instrumentation and coarse grain sampling so as not to increase the execution overhead and the amount of data to be analyzed, while obtaining highly detailed information. To achieve this objective we first extend the MPItrace tracing package with sampling mechanisms, and then apply a process that improves the granularity of the obtained samples, if needed.

MPItrace is a trace-based instrumentation tool that collects performance information about parallel applications. The resulting combination provides more data of the application by gathering information from uninstrumented regions of code, independent of the application structure.

The rest of this paper is structured as follows: Section 2 provides information about the implementation of the sampling mechanism combined with the MPItrace instrumentation package. Section 3 introduces the methodology used to analyze traces containing information gathered by the instrumentation and the sampling mechanisms. This methodology is evaluated in section 4. Section 5 gives an overview of related work. Finally, we plot the conclusions and future trends in Section 6.

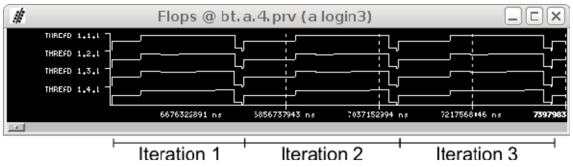
2 Instrumentation and Sampling Mechanisms

MPItrace is an instrumentation tool that collects performance information of parallel MPI applications automatically through the MPI profiling interface [11]. It also provides information of OpenMP constructs and some pthread calls by replacing the references to the runtime found in shared libraries by using dynamic library interposition mechanisms and allows the user to add his owns events in the resulting trace manually.

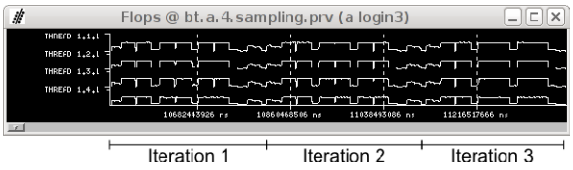
The sampling mechanism added to the MPItrace instrumentation package is built on top of PAPI [6] because it provides a flexible and precise way to determine the sampling rate [14]. This library provides a method called `PAPI_overflow` to setup the sampling frequency based on a hardware counter and a threshold. Once the selected counter reaches the given threshold, it triggers an exception that is captured by PAPI and forwarded to a callback function. The programmed callback collects processor performance metrics and the address within the process that was being executed. We consider henceforth a *sample* as the set of data containing the performance metrics on a specific time and the process address.

The utility of this framework, which extends with sampling the typical traces obtained with MPItrace, is shown in Figure 1 of the Paraver visualization tool.

Paraver represents on the y-axis the application's threads while the x-axis represents time. On both images, Paraver is configured to draw the number of floating point operations. In the left image, the number of floating points operations are drawn at instrumentation points, whereas in the right image the number of floating point operations are drawn at sampling points using a fine resolution. It is noticeable that the Figure 1(b) plots performance information not only on the default instrumented points (MPI routines in this case), but along the iteration. The figure on the right improves the quality of the analysis by detailing the achieved floating operation performance in a finer way than the figure on the left. Details that can be observed in the image on the right are: a) variance at the very beginning and end of each iteration, and, b) that every iteration has three separated regions of code that deliver a high number of floating point operations, and that each of those three regions has a short and sharp decrease of performance in the middle of the region.



(a) Paraver windows showing the automatic information gathered by the instrumentation showing the variation of Mflops



(b) Paraver view showing the variation of Mflops using a trace with data captured by instrumentation and sampling

Fig. 1. Paraver windows showing information captured by instrumentation with and without sampling mechanism

3 Analysis Methodology

The procedure we follow to characterize applications focuses on selecting the most time consuming routines, avoiding library internals and inline or intrinsic routines. We want to provide detailed performance metrics, the number of invocations, the time they need to execute and their variation across the iterations for the selected routines. This methodology requires the user to know in advance which routines are important, possibly by previously analyzing the application with profilers. We know that this is a limitation of the methodology and we are currently working on a more automatic way to make it easier to apply. In this paper we aim to increase the details of the selected routines by using both instrumentation and sampling mechanisms.

The first step consists of obtaining a trace using the MPItrace instrumentation package with sampled information. Then the analyst will evaluate the application afterwards using Paraver. This tool provides quantitative and qualitative analysis of the traces using timelines and histograms and also contains a set of predefined windows to conduct the analysis and facilitate the characterization of the target application. Among them, there are several related windows that provide information for user routines and performance metrics.

As the knowledge about the target application may be limited, choosing a proper sampling rate to detail the application enough is a blind process. In case the selected rate results in too few metrics scattered all over the execution, they can be combined to produce highly detailed representative areas using a process that we call *folding*.

The folding process is suitable for applications based on iterative methods, which is a representative part of the applications found in high performance computing environments. For the rest of applications, a new trace could be generated with finer sampling resolution.

To proceed with the combination, the process can work at three different granularities: full iteration, instrumented user routines and running bursts (*i.e.* intervals between MPI calls).

The full iteration granularity provides a detailed view of a single iteration. Using this behavior each sample that is run within the main application loop is treated equally, independent from where it was emitted. The second granularity focuses only on user code that has been instrumented by the analyst and ignores the rest. Finally, the last option allows working separately on different intervals delimited by MPI calls. This granularity grants working only on user code and its results are not interfered by MPI calls.

We detail the folding process for full iterations in section 3.1. Applying the folding process to the rest of granularities just implies splitting the iteration every time an instrumented function or a MPI call is found.

Concerning the implementation, little modifications of the source code are required to perform the iteration folding. The user just needs to instrument the main loop so as to delimit the beginning and end of each iteration and gather the performance counter values at these points, and optionally, instrument the interesting user functions.

3.1 Folding Iterations

If the chosen sampling rate is too coarse, the performance metrics will not be precise enough to characterize any region of code. We are interested in how to obtain precise information under these circumstances. Our objective is to build a synthetic trace to provide information of the behavior of an iteration per MPI task with high precision. The subsequent analysis procedure must only target the resulting iteration and then extrapolate the results to the rest. The ability of treating each task independently allows spotting performance differences across tasks.

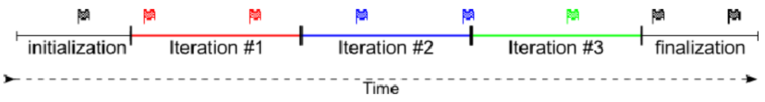
To properly obtain fine grained information relative to the performance counters we apply a two-step procedure within each task: migration of samples into a single iteration and hardware counter interpolation. To proceed with the last step, the user must indicate the desired number of output samples. Requesting more samples yields more detailed results.

Hardware Counters Folding. We define *folding* an iteration (*source*) into another (*destination*) as the operation that migrates samples from source to destination preserving the offset of source iteration and converts its performance counter values so as they are relative to the beginning of the iteration they originally belong. Figures 2(a) and 2(b) illustrate an example. The top figure shows a timeline of a program that runs 3 iterations of its main loop and the bottom figure shows the same timeline after folding the second and third iterations into the first one.

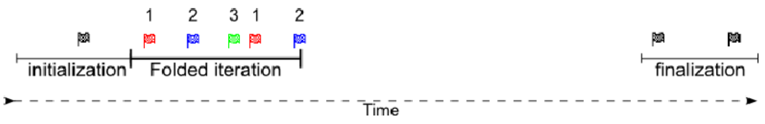
Considering applications that have a regular and iterative control and data flow and dedicated systems with low external interferences (such as preemptions, interrupts, network and memory contention, *etc...*), the resulting set of iterations conform to an ergodic system. This is to say, any iteration matches up with the typical iteration and, thus, samples can be migrated from one to another without altering the gathered values of the performance metrics.

In practice, executions are non deterministic. Even using dedicated systems, applications face different interferences each run, which may result in slight performance variations on each iteration. Considering the duration of the iteration as a normally distributed variable, we can safely remove those iterations that last more than twice the standard deviation. After this removal, we are still working with a representative set of iterations because we are keeping the iterations that their duration are within the interval of confidence of the 95% of the whole samples.

Concerning the output resolution, folding all the iterations into a single one results in a single iteration that contains all the samples scattered across



(a) Samples in full application run with 3 iterations



(b) Samples in folded iterations

Fig. 2. Flags in those figures represent points where information is located before (a) and after (b) folding all iterations. The superindices and colors of flags on Figure (b) show which was their original iteration.

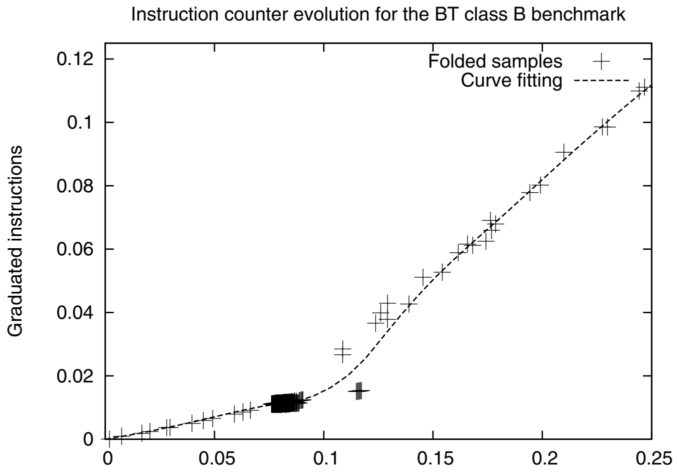


Fig. 3. Evolution of graduated instructions for the BT benchmark within an iteration

iterations, or in other words, the resolution of the folded iteration is proportional to the number of iterations folded.

Hardware Counters Interpolation. Once the hardware counter metrics have been folded into a single iteration, we take as many equidistant samples from this set of data as the user requested. Since hardware counters information is punctual in time, we build a continuous function that closely fits the set of data to estimate all the hardware counters values across the whole folded iteration.

This process is known as interpolation and in order to apply it we explored several approaches: polynomial fitting, Bézier curves [5] and Kriging [21] interpolation. Polynomial fitting requires choosing the grade of the polynomial and this cannot be done independently from the data. In addition, choosing a low order polynomial will give soft but inaccurate fitting, while a high order polynomial will fit better but will result in big fluctuations. Bézier curves do not require additional data but the points themselves. Bézier fitted well on our tests but on the stationary points. The Kriging interpolation is a general version of the Bézier curve typically used in contouring. Kriging algorithm works with the sample points plus some interpolation parameters (including fitting strictness). After some tests, we found a typical combination of parameters that fitted the samples well even in stationary points.

Although performance counters are monotonically increasing in a single iteration, it may not be completely monotonic when considering the whole set of samples. Consider the case shown in Figure 3. This figure plots the graduated instructions across time. There are several points with more graduated instructions than near points in the future due to variations of performance counters values on each iteration. Even though this effect was previously minimized by the outlier removal and by the tendency of the interpolation function to ignore spurious values, they can still appear. Whenever this happens, they are just ignored.

This process must be repeated for each task present in the tracefile and concludes reintroducing the values of the interpolated performance counters into the trace.

4 Case Study

This section shows how the folding process and the combination of instrumented and sampled performance information contributes to improve the detailed analysis of real applications.

Table 1. Characteristics of the system used for the evaluation

Experimental systems characteristics	
Processor family	Intel Itanium 2
Processor frequency	1.6 GHz
PAPI version	3.6.2
Linux kernel	2.6.16.46-0.12
Compilers (C/Fortran)	icc and ifort 11.0

Table 2. Setup of the different experiments

Application	NAS bt.B	Alya
Number of processors used	16	4
Sampling frequency (in million cycles)	50	1000
Average duration per iteration	183ms	18.2s
Samples per iteration	5 - 6	28 - 29
Number of timesteps	200	100
Runtime overhead	1.5%	3%

To perform our analysis using the folding process we choose bt.B from the NAS MPI Parallel Benchmark Suite 3.2 [3], and Alya [12], a computational mechanics simulator that is typically run in our production environment. Table 1 describes the characteristics of the system used in this study and table 2 provides information about the setup of the different experiments. The overhead row in table 2 comprises the overhead of the sampling mechanism plus the MPI instrumentation and the manually added events used to identify the iterations.

We request a thousand samples in the target folding iteration regarding the performance metrics. This is equivalent to set a sampling frequency 200 and 35 times higher in bt.B and Alya respectively with the proportional increase of overhead. In addition to that, we use the user function granularity of the folding process in order to improve the understanding of the results by providing detailed information of representative functions.

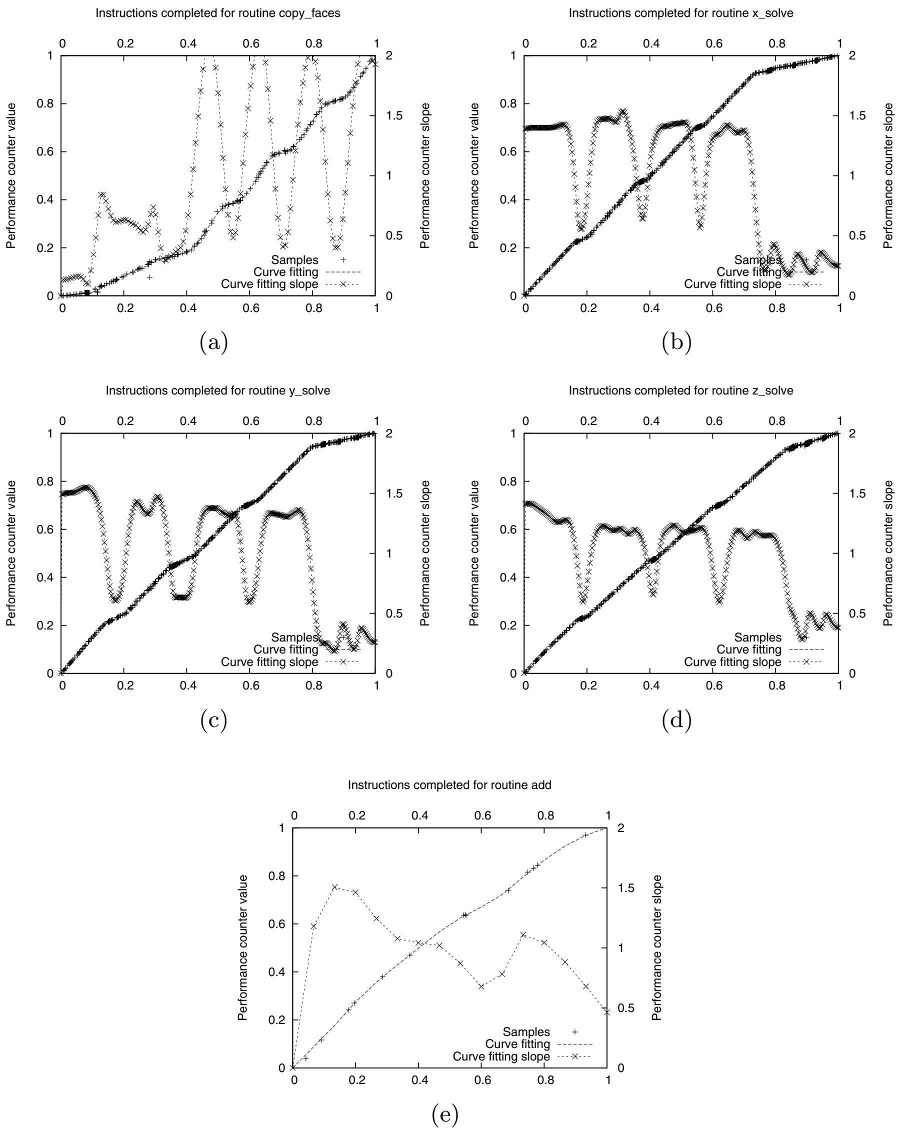


Fig. 4. Evolution of completed instructions for the 1st task in bt.B benchmark in the instrumented routines after applying the folding mechanism

4.1 NAS bt.B

Previous experience on this benchmark has shown that the interesting routines are: `copy_faces`, `x_solve`, `y_solve`, `z_solve` and `add`.

Figures plotted in Figure 4 show the evolution of the completed instruction performance counter among each of the five routines in the first task. The x-axis represent the time (normalized from 0 to 1) in the routine, the left y-axis is used to show the value of the counter (normalized from 0 to 1) within the routine and the right y-axis is used to range the slope of the interpolation. Each plot presents three types of information:

- Samples gathered during instrumentation and folded into the chosen iteration and within the user functions that were taken. Samples are shown by crosses.
- Interpolation of the gathered samples. It is shown by a segmented line.
- Slope of the interpolation of the gathered samples. It is shown by a segmented line with crosses.

The slope of the interpolation facilitates the location of hot-spots (or even cold-spots). For example, in Figure 4(a) three different behaviors exist. First, a small section, which lasts about the first 10% of time, aggregates completed instructions very slowly. Then comes a region with a higher instruction completion rate. And finally, a long region that has four peaks separated by valleys. Looking at the source code of the routine, we find that this behavior corresponds with the execution of a loop in the subroutine `compute_rhs` that is executed four times.

Routines `x_solve`, `y_solve` and `z_solve`, which are shown in Figures 4(b), 4(c) and 4(d) respectively, present a very similar pattern. There is a region that lasts approximately the 80% of time with four peaks that accumulate more than the 90% of the completed instructions. The remaining count of completed instructions come from the remaining 20% of time.

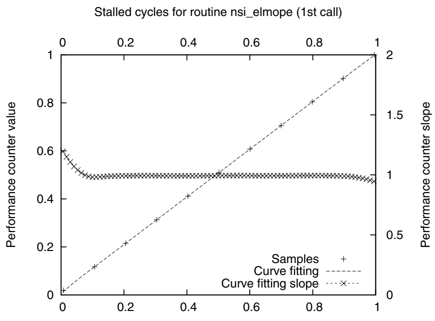
Finally, Figure 4(e) shows the behavior of the `add` routine. It starts by accumulating a high number of completed instructions and then it decreases slowly. At 75% of the routine time it increases again to decrease at the end of the routine.

4.2 Alya

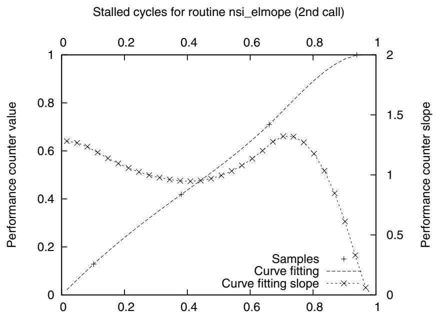
Alya developers helped us locating the most time consuming routines. These routines are: `nsi_elmope`, `gmrpls` and `cgrpls`. We focus on the second task because the parallel computation is done in all tasks except the first, which synchronizes the parallel computation.

Figures plotted in Figure 5 show the evolution of the stalled cycles performance counter among each of the three routines in the second task. The stalled cycles performance counter provides information of the amount of time that the CPU has been waiting for resources to execute an instruction. Such counter is useful to locate code region with bottlenecks that prevents the application going at full speed.

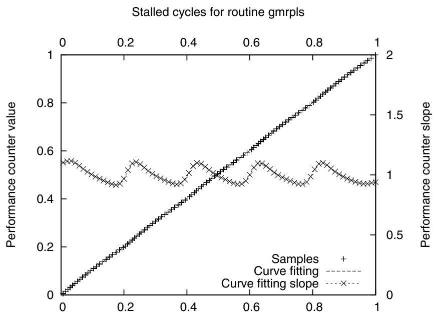
Although there are three instrumented routines, one of them is executed twice in an iteration and thus we present two different plots for it (one for each execution). As seen in the BT example, x-axis represent the time in the routine, the



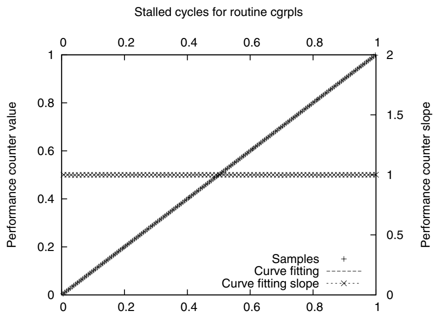
(a)



(b)



(c)



(d)

Fig. 5. Evolution of stalled cycles for the 2nd task in the Alya in the selected routines after applying the folding mechanism

left y-axis ranges the value of the counter within the routine and the right y-axis delimits ranges the slope of the interpolation. Each plot presents three types of information: samples gathered, interpolation and the slope of the interpolation.

First of all, it is noticeable that the two calls of `nsi_elmope` behave differently. This routine does the assembly of the continuity and momentum matrices. Regarding its performance, the first call plot is shown in Figure 5(a). The plot presents a steady behavior across the execution except for the margins. However, the second call, which is shown in Figure 5(b), accumulates most of the cycles stalled at the beginning of the run and it suddenly decreases at the 80% of the routine after a peak at that position.

The `gmrpls` routine implements an iterative gmres solver for nonsymmetric matrices. We see in Figure 5(c) that the slope of the evolution of the performance counter behaves as a wave with small amplitude and with a wavelength of 20% of the routine duration.

Finally, the plot in Figure 5(d) shows the performance behavior of the the routine `cgrpls`, which implements a conjugate gradient method. It behaves steadily across the whole execution.

5 Related Work

In this section we summarize the approaches done to combine both instrumentation and sampling mechanisms in the performance analysis area, and how they differ with our work.

The widely known gprof [10] exploits both sampling and instrumentation mechanisms to emit functions call count and estimated spent time. Gprof requires the application to be compiled with a special flag that is responsible for instructing the compiler to add counting monitors in the user routines. Gprof also uses sampling mechanisms to attribute time to the user routines during the execution. The visualization tool echoes to the standard output the fraction of time spent when a routine directly called another one and the number of invocations for each routine. The main difference with our work is that gprof is a performance tool based on profiling whereas our tool is based on tracing. Gprof just provides summaries for simple and concrete function metrics. The solution we propose combines information gathered by tracing and instrumentation to generate a trace with timestamped details that makes the analysis more detailed and precise.

The Sun Studio Performance Analyzer [13] comprises a set of tools for collecting and viewing application performance data using tracing and profiling mechanisms. Its collecting tool is able to trace information relative to synchronization calls, heap allocation and deallocation, OpenMP constructs, MPI routines, data-race and deadlock detection, and counting the number of times each instruction was executed. Furthermore, it uses sampling clock-based or hardware counter overflow mechanisms to profile the target application. Its visualization tool is a multifunctional window that presents the data collected in a wide variety of flavors, including: flat routine profile (similar to *gprof*), calling and called routines, link to source and disassembly, and a timeline window. It exploits the sampling to accumulate the execution time for different routines, and, identify which parts of the user program are responsible for cache or floating point inefficiencies. Although being a powerful set of tools, it needs to collect again the performance data if the results are not detailed enough. This is not an issue for the solution we propose if the application matches a set of requirements described in section 3.1.

Azimi, Stumm and Wisniewski present in [?] an online performance analysis tool that gathers counter values periodically or after a designated number of hardware counter occurrences. This tool presents the evolution among the selected counters in a timeline and provides accurate information on which micro-architecture components are stressed using a model called Statistical Stall Breakdown. To provide information for all performance counters it multiplexes counters taking advantage of the underlying functionalities provided by the operating system. Although they offer some instrumentation capabilities, their work is just focused on the sampling mechanism to characterize the whole system and not applications.

SimPoint [20] and SMARTS [23] use sampling in a different context so as to accelerate detailed micro-architecture simulations. Their primary goal is to reduce long-running applications down to tractable simulations. The authors of both frameworks met this goal by statistically sampling the instruction mix of a running serial application to determine where the application spent time. They combine instrumentation and sampling to collect a sequence of instructions each time the sampling mechanism fires up. The result they obtain is a collection of instruction traces related to each sampling point. These traces will be simulated independently to get detailed performance on the simulated micro-architecture. Our approach, however, combines performance information gathered at different timesteps into a single timestep and the coarse grain sampling produces low overhead penalty in the application run.

A tracing package with some sampling capabilities is presented in [16]. Its functionality is closely related to SimPoint but combining pSiGMA [18] and DynInst [7] instrumentation. Their approach works identifying the timesteps of the application using special instrumentation calls and fully instrumenting the application using the SiGMA toolkit. A DynInst-based tool instruments the special calls added to check whether the executed timestep matches a list of timesteps given by the user to enable or to disable the pSiGMA instrumentation. The fully instrumentation inflicts large overhead penalty when gathering performance data, although it can be greatly reduced by sampling a small set of iterations. Our approach, however, is to take few samples along the application execution and then construct an ideal iteration from the data collected.

6 Conclusions and Future Work

We have explored the possibility of combining both sampling and instrumentation mechanisms to generate more detailed traces.

Our main contribution is the design and implementation of a process called *folding* that provides detailed performance information using coarse grain sampling. It provides detailed performance information at three different levels: iteration, user routines and running bursts. It is suitable for applications based on iterative methods, which is a representative part of the applications found in HPC environments. For the rest of applications, a finer sampling resolution should be chosen to obtain a trace with more details.

To demonstrate its utility we have extended the MPItrace instrumentation package with a sampling mechanism using hardware counters to record performance information across the whole execution. The combination of instrumentation and sampling produces traces containing performance information from both instrumented and uninstrumented regions of code.

The additional performance information provided by the sampling, and the ability to project it into representative areas, brings a new methodology into play that has proven useful for the detailed analysis of real applications. In particular, we have shown detailed performance analysis for the representative routines of the NAS BT benchmark and the Alya application.

Finally, we believe that some of issues in the methodology are still open.

First, provide a way to automatically determine which are the interesting and representative user routines to be analyzed without using a profiler. Samples could also store the stack trace, in addition to the performance counter values and the address of the instruction that triggered the sample, in order to provide information of the executed routines.

Second, automatically identify the application iterations structure using periodicity or application structure detectors like [8] and [9]. These detectors work directly with Paraver traces and the reported information could be used as input of the folding mechanism to delimit the regions to be folded instead of adding manually events into the application source code.

Finally, we would like to study the impact of the sampling rate and the number of sampled timesteps on the folding results.

Acknowledgements

We would like to acknowledge José María Cela and Pierre Lafortune for their insightful comments on contouring algorithms and the implementation of the Kriging algorithm. This work is granted by the IBM/BSC MareIncognito project and by the *Comisión Interministerial de Ciencia y Tecnología* (CICYT) (contract no. TIN2007-60625).

References

1. Intel trace collector and analyzer, `http://www.intel.com/cd/software/products/asmo-na/eng/306321.htm`
2. MPItrace instrumentation package, `http://www.bsc.es/plantillaA.php?cat_id=492`
3. NAS parallel benchmark suite, `http://www.nas.nasa.gov/Resources/Software/npb.html`
4. Azimi, R., Stumm, M., Wisniewski, R.W.: Online performance analysis by statistical sampling of microprocessor performance counters. In: ICS '05: Proceedings of the 19th annual international conference on Supercomputing, pp. 101–110. ACM, New York (2005)
5. Bézier, P.: Numerical Control. Mathematics and Applications. John Wiley and Sons, London (1972) Translated by: Forrest, A.R., Pakhurst, A.F.
6. Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.: A portable programming interface for performance evaluation on modern processors. Int. J. High Perform. Comput. Appl. 14(3), 189–204 (2000), `http://icl.cs.utk.edu/papi`
7. Buck, B., Hollingsworth, J.K.: An API for runtime code patching. Int. J. High Perform. Comput. Appl. 14(4), 317–329 (2000), `http://www.dyninst.org`
8. Casas, M., Badia, R.M., Labarta, J.: Automatic Structure Extraction from MPI Applications Tracefiles. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 3–12. Springer, Heidelberg (2007)
9. González, J., Giménez, J., Labarta, J.: Automatic Detection of Parallel Applications Computation Phases. In: IPDPS'09: 23rd IEEE International Parallel and Distributed Processing Symposium (2009)
10. Graham, S.L., Kessler, P.B., Mckusick, M.K.: Gprof: A call graph execution profiler. In: SIGPLAN '82: Proceedings of the 1982 SIGPLAN symposium on Compiler construction, pp. 120–126. ACM, New York (1982)

11. Hempel, R.: The MPI standard for message passing. In: HPCN Europe 1994: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, London, UK, vol. II, pp. 247–252. Springer, Heidelberg (1994)
12. Houzeaux, G., Vázquez, M., Grima, R., Calmet, H., Cela, J.M.: Experience in parallel computational mechanics on marenostrum (2007)
13. Itzkowitz, M.: Sun studio performance analyzer
14. Moore, S.V.: A comparison of counting and sampling modes of using performance monitoring hardware. In: Sloot, P.M.A., Tan, C.J.K., Dongarra, J., Hoekstra, A.G. (eds.) ICCS-ComputSci 2002. LNCS, vol. 2330, pp. 904–912. Springer, Heidelberg (2002)
15. Nagel, W.E., Arnold, A., Weber, M., Hoppe, H.C., Solchenbach, K.: VAMPIR: Visualization and analysis of MPI resources. Supercomputer 12(1), 69–80 (1996)
16. Odom, J., Hollingsworth, J.K., DeRose, L., Ekanadham, K., Sbaraglia, S.: Using dynamic tracing sampling to measure long running programs. In: SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, p. 59. IEEE Computer Society, Los Alamitos (2005)
17. Pillet, V., Labarta, J., Cortés, T., Girona, S.: Paraver: A tool to visualize and analyze parallel code. Transputer and occam Developments, 17–32 (April 1995), `http://www.bsc.es/plantillaA.php?cat_id=485`
18. Sbaraglia, S., Ekanadham, K., Crea, S., Seelam, S.: pSIGMA: An infrastructure for parallel application performance analysis using symbolic specifications. In: Proceedings of EWOMP'04 (October 2004)
19. Shende, S.S., Malony, A.D.: The TAU parallel performance system. Int. J. High Perform. Comput. Appl. 20(2), 287–311 (2006)
20. Sherwood, T., Perelman, E., Hamerly, G., Calder, B.: Automatically characterizing large scale program behavior. SIGOPS Oper. Syst. Rev. 36(5), 45–57 (2002)
21. Trochu, F.: A contouring program based on dual Kriging interpolation. Engineering with Computers 9(3), 160–177 (1993)
22. Wolf, F., Mohr, B.: Kojak - a tool set for automatic performance analysis of parallel applications. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 1301–1304. Springer, Heidelberg (2003); Demonstrations of Parallel and Distributed Computing
23. Wunderlich, R.E., Wenisch, T.F., Falsafi, B., Hoe, J.C.: Smarts: accelerating microarchitecture simulation via rigorous statistical sampling. In: ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture, pp. 84–97. ACM, New York (2003)

Automatic Performance Analysis of Large Scale Simulations*

Shajulin Benedict[1], Matthias Brehm[2], Michael Gerndt[1], Carla Guillen[2], Wolfram Hesse[2], and Ventsislav Petkov[1]

[1] Technische Universität München
Fakultät für Informatik I10, Boltzmannstr. 3, 85748 Garching, Germany
[2] Leibniz Computing Centre, Hochleistungsrechnen, Boltzmannstr. 1, 85748 Garching, Germany

Abstract. Developing efficient parallel programs for supercomputers is a challenging task. It requires insight into the application, the parallelization concepts, as well as the parallel architectures. Performance analysis tools such as Periscope, an automatic performance analysis tool currently under development at Technische Universität München, help the programmer in detecting performance bottlenecks. The goal of the ISAR project is to enhance the existing Periscope research prototype and deliver a production version. This paper focuses on the evaluation of Periscope's main features based on two large scale simulation codes.

Keywords: Performance analysis, program tuning, program transformations.

1 Introduction

Periscope [2] is an automatic performance analysis tool that searches for predefined performance properties which are based on measurements during program execution. In contrast to other tools, it is based on a formal specification of performance properties and applies an online search via a network of analysis agents while the application is running. This approach guarantees the applicability to large scale simulations, i.e., simulations running on large number of processors for a long time.

While many HPC applications are well tuned with respect to parallel execution, they use only a small percentage of the compute core's peak performance. This gap is very significant and will probably become even more important as the processor's systems structure gets more and more complex. Therefore, Periscope features performance properties that identify inefficient usage of the processors, especially of the memory hierarchy. This support is based on the stall cycle counters of the Itanium 2 processor used in the Altix supercomputer at the Leibniz Supercomputing Centre (LRZ). The results shown in this paper are focusing on the single node performance.

* This work is partially funded by BMBF under the ISAR project, grant 01IH08005 and the SILC project, grant 01IH08006.

In this paper, we evaluate Periscope based on two large scale simulations, the plasma physics code GENE and the geophysics code SeisSol.

The analysis of the two large scale simulation codes are part of the BMBF research project ISAR[1], that is funded until 2011. Within this project, the current research prototype of Periscope will be productized. In addition, an HPC system monitoring tool will be developed based on Periscope that allows the compute center to monitor all applications running on large scale supercomputers to detect inefficient applications that could profit from further tuning.

The rest of the paper is organized as follows. Section 2 gives an overview of Periscope and Section 3 presents related work. Section 4 discusses the techniques used in Periscope to guarantee scalability. Sections 5 and 6 introduce the two application codes and present the results for each of the code.

2 Periscope

Periscope is a scalable automatic performance analysis tool currently under development at Technische Universität München. It consists of a frontend and a hierarchy of communication and analysis agents (Figure 1). One analysis agent, i.e., a leaf of the agent hierarchy, has one or more application processes to analyze. Each of the analysis agents searches autonomously for inefficiencies in a subset of the application processes.

The application processes are linked with a monitoring system that provides the Monitoring Request Interface (MRI). The agents attach to the monitor via sockets. The MRI allows the agents to configure the measurements; to start, halt, and resume the execution; and to retrieve the performance data. The monitor currently only supports summary information.

Periscope starts its analysis from the formal specification of performance properties on the User Region. Before the instrumentation of the code, the user has to define the region of interest, i.e., the User Region. Further analysis is made to detect other regions, such as loop regions, call regions and sub regions. The specification determines the condition, the confidence level, and the severity of performance properties. The severity is the percentage of the time lost due to the problem the property describes. We examine relative values of time as percentage rather than absolute values. An example of a typical property is time lost due to stall cycles. Its severity is therefore computed as:

$$Severity = (StallCycles/PhaseCycles) * 100\%$$

and associated condition of this property depends on the a defined threshold:

$$Condition = Severity > Threshold$$

This means that the condition can only be true or false. Besides properties for MPI and OpenMP, Periscope has properties for detecting inefficient single-node

[1] `http://www.in.tum.de/en/forschung/verbundprojekte/clusteraktivitaeten/isar.html`

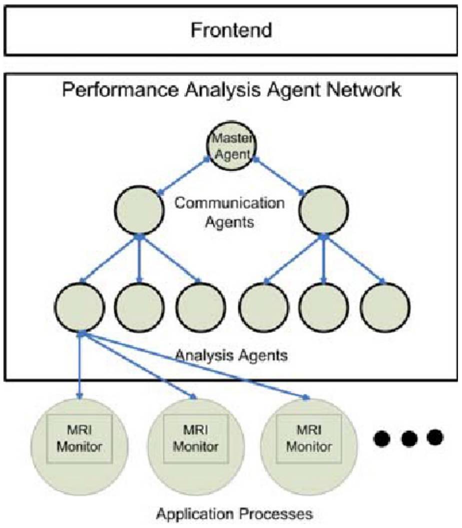


Fig. 1. Periscope consists of a frontend and a hierarchy of communication and analysis agents. The analysis agents configure the MRI-based monitors of the application processes and retrieve performance data.

or better single-core performance. The search is performed according to a search strategy selected when the frontend is started [3]. At the end of the local search, the detected performance properties are reported back via the agent hierarchy to the frontend.

All the current properties in Periscope give the percentage of execution time lost by this property. This allows to rank all the found properties and let the programmer start optimizing the code for the worst one. Properties are implemented in Periscope in form of C++ classes.

Changes to previous strategies include improvements performed in *MPI* Strategy. A new strategy available called *All* was developed and this includes all the previously defined strategies, namely *Stall Cycle Analysis* and *MPI*.

3 Related Work

Some well known performance analysis tools for parallel systems are already available. The most notable ones are Paradyn, TAU, Vampir, KOJAK, SCALAS-CA, and mpiP.

Paradyn [4,5] was the first tool that automated performance analysis. Its Performance Consultant guides instrumentation and searches for bottlenecks based on summary information during the program's execution. TAU [6] is a comprehensive environment supporting trace-based and profiling-based performance analysis. It performs an offline analysis and provides a wide variety of graphical and text-based displays.

KOJAK [7] also provides trace-based analysis of parallel applications. It includes Expert, a component that automatically deduces performance properties from the trace files. SCALASCA [1] can be seen as a more scalable version of KOJAK. KOJAK's search for performance properties is now done in a parallel post-processing step on the same CPUs which were used for the execution of the application.

4 Concepts for Scalability

The current version of Periscope runs on the Altix supercomputer installed at the Leibniz Supercomputing Centre in Munich.

The Altix supercomputer is a ccNUMA system with 19 partitions, each of which has 256 Itanium 2 dual core processors with a peak performance of 12.8 GFlops. The NUMA link4 communication network has a fat tree topology in the partitions and a 3D torus topology across the partitions. Each NUMA link has a peak communication bandwidth of 3.2 GB/s in each direction. Periscope can be used in other architectures, namely AMD64, IA32 with Linux and IA32 with Windows.

The application and the agent network are started through the frontend process. It analyzes the set of processors available, determines the mapping of application and analysis agent processes, and then starts the application and the agent hierarchy. After startup, a command is propagated down to the analysis agents to start the search.

To be able to analyze large test runs, Periscope has to support batch jobs. The specific problem here is that jobs with many cores will be spread over multiple partitions. Periscope's frontend gathers information about all the partitions with cores assigned to the job before it starts optimizing the mapping of application processes and analysis agents. The computed mapping is then enforced via the `dplace` command and appropriate mapping files for each partition.

For the communication among the agents and between the agents and the application we use sockets. For small scale runs the socket range could be defined before starting the search. For large scale runs, there are always blocked sockets and thus, we had to modify the startup process to enable dynamic selection of sockets. Prior to supporting large runs, Periscope had as a baseline runs with less than 64 processes.

The next modification required was to reduce access to the registry. The registry is a book keeping mechanism which keeps track of network data such as where the application and the agents are executing and via which port they can be contacted. Since the registry is handling only a single request at a time, the startup of thousands of application processes and the requests of the agents for accessing this information became a severe bottleneck. Therefore the application now collects all the information about the processes in a master process which then transfer it into the registry. The identifiers returned from the registry for each process are now distributed via MPI too.

When getting the properties found for over 1000 processes, it became obvious that the user will not be able to look at all the results. Therefore, we started to investigate using clustering techniques to identify classes of processes behaving in the same way. This technique is currently introduced into the Periscope GUI which will become available soon.

5 GENE

The Gyrokinetic Electromagnetic Numerical Experiment (GENE) of the Max Planck Institute for Plasma Physics in Garching, a large-scale simulation tool, supports the magnetic confinement fusion community involved in nuclear reactors research to solve the problem of plasma microturbulence. GENE aims at solving the non-linear gyrokinetic equations in a 5-dimensional phase space to identify the turbulence in magnetized fusion plasmas.

GENE consists of 47 source files with 16,258 lines written in Fortran 95 and parallelized with MPI. This section shows results obtained from the experiments conducted with Periscope on this code. The large-scale runs were performed by submitting it in batch jobs to the ALTIX machine. The results reveal the properties in different code regions of GENE and the single node performance for a 1024 processor run.

We did test runs of GENE with 1, 8, 16, 32, 64, 128, 256, 512 and 1024 processors. The test runs were started via Periscope's frontend. It first starts the application and, after the processes registered with the registry, starts the analysis agents. For small-scale runs, such as, 1, 8, or 16 processors, a single analysis agent and a master agent was used for analysis. However, for large-scale

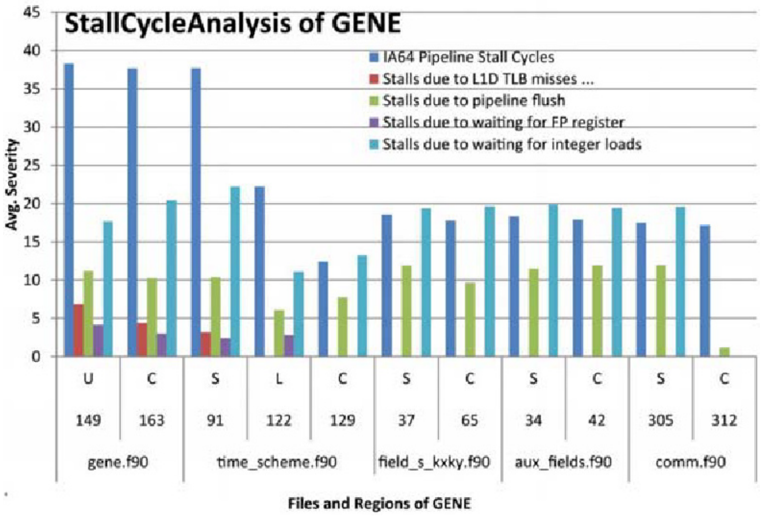


Fig. 2. Properties in different code regions of GENE. You see the file name, the line number, and the region type (U for user , L for loop, S for subroutine, C for call).

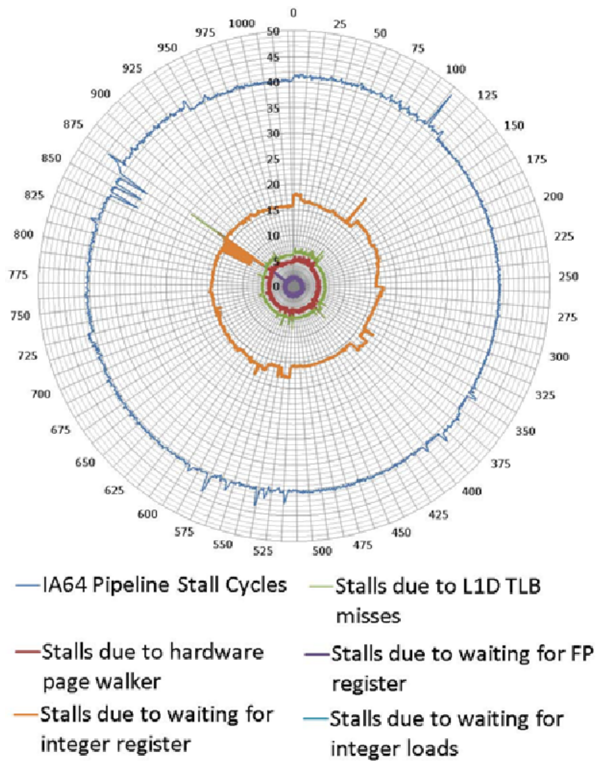


Fig. 3. Single node performance of GENE for 1024 processor run in ALTIX

runs, the processes were started on multiple partitions of the Altix, and hence multiple analysis agents were used.

The main program, gene.f90, has three main steps such as i) it reads the parameters, ii) sets the initial condition and iii) enters into the explicit time loop before it ends. The user-region indicating the phase for the incremental analysis covers the code for one time step. The explicit time loop runs iteratively to solve the gyrokinetic partial differential equation.

Periscope identified the properties, namely, stalls in the processor pipeline, stalls due to L1D TLB misses, stalls due to pipeline flush and much more. Figure 2 shows the mean severity for all processes for the most critical program regions. It is interesting that the code suffers more from stalls due to integer loads than from stalls due to floating point registers.

As an example, the following graph (Figure 3) depicts the single node performance of GENE code on considering 1024 processor run. In this graph, processor 112 had an unexpected peak of 13 severity points from its average for IA64 pipeline stalls, and processor 868 had 19 severity points for Stalls due to L1D TLB misses. It can be revealed that, processors 844 to 853 had comparatively good performance due to less severity than other processors.

6 SeisSol

SeisSol [8] is an application for large-scale simulations of seismological activity, developed by the Department of Earth and Environmental Sciences at the Ludwig-Maximiliam-Universität. It provides wave propagation solutions in an elastic medium in 3D with geometrically complex domains. Several real world scenarios have already been modelled with this solver. The computational effort of solving two scenarios was analyzed with Periscope and its results are presented.

The main calculations involve solving linear systems of equations along the discretized cells. The parallel version of SeisSol starts from an already partitioned domain. SeisSol takes as input the coordinates of a meshed domain along with material parameters and boundary conditions. The mesh is preprocessed to have different configurations related to the number of MPI tasks. The MPI tasks read the data from an input file and process each block in parallel allowing boundary communication at each subdomain.

SeisSol has four main program sections. The first one, data initialization, reads input files required for running the simulation. Most of the calculations are done in the second section, going over multiple time steps and numerous mesh cells to solve large systems of equations. The User Region was therefore defined within the file calc_seissol.f90 which contains the main loop. The other two sections, analysis of results and final output, were not analyzed given that they don't contain several nested loops with heavy calculations.

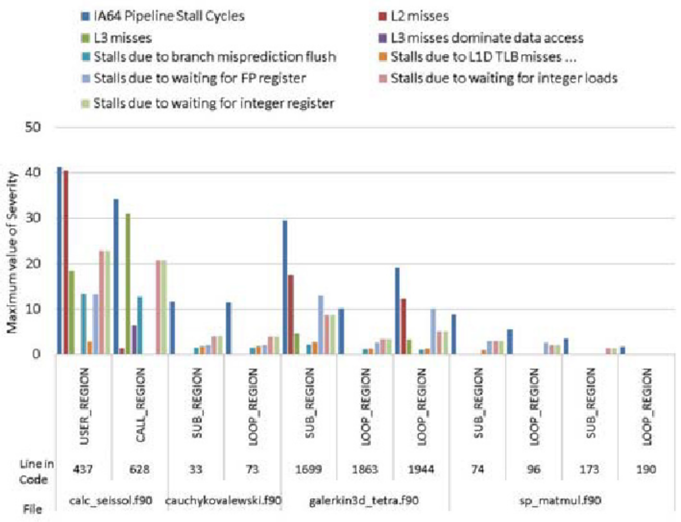


Fig. 4. Graph showing maximum value of the severity of properties found. One analysis agent was used to analyze SeisSol on 64 processors. The first line of the region inside the code is shown.

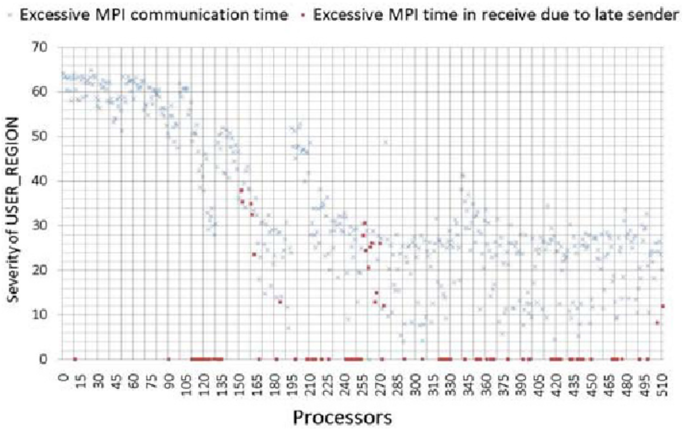


Fig. 5. Maximum severity of the properties found with Periscope agent hierarchy on 512 processors

The instrumented executable from SeisSol ran on configurations 4, 8 and 64 processors with one analysis agent from Periscope. Furthermore, configurations with 256 and 512 processors were also tested using the Periscope hierarchy, which includes the frontend, highlevel agents and analysis agents.

The properties found using 64 processors and the *Stall Cycle Analysis Breadth First* strategy are presented in Figure 4. The experiment showed that the CALL_REGION, which is a call to an MPI all-reduce, has several pipeline stall cycles and L3-Cache misses, with an average of 32% of loss of computation time. Given that the pipeline stall cycles are at an MPI all-reduce call, we infer that some processors are idle waiting for the entire group communication to be finished which thus might indicate that load imbalance could be a potential problem among partitions. L2-Cache misses drew our attention to the SUB_REGION and LOOP_REGION of galerkin3d_tetra.f90. This section of the code contains calls to solvers with matrix multiplication. In order to optimize these regions we can think of data locality optimizations to avoid cache misses.

The hypothesis that the MPI all-reduce call is creating stall cycles due to load imbalance of the partitions was further investigated with Periscope. The results for the strategy *All* and a configuration for 512 processors are shown in Figure 5. The severity of excessive MPI communication time shows that 65% of the time is lost in one or more processors waiting for the blocked communication.

7 Summary and Conclusions

Periscope has proven to be scalable with both applications, GENE and SeisSol. The tests executed with the applications were run on up to 1024 processors on GENE and 512 on SeisSol. The configuration data sets for larger scale runs in both applications were not available. Nevertheless, a scalability of 2048 processors was achieved with Periscope on a test code.

Advances in Periscope included the improvement of the instrumentation process. The applications can now be instrumented using a script called from the code's makefile, thus simplifying the instrumentation step. Several problems limiting the scalability on the side of Periscope were solved and larger scale runs are possible. Successful tests were executed to base the communication on MPI as well. A new strategy available called *All* was developed and this includes all the previously defined strategies, namely *Stall Cycle Analysis* and *MPI*.

As part of the three year project ISAR, Periscope will also be ported to other machines, such as the Blue Gene P. This requires to change the current implementation of the communication within the agent network and between the analysis agents and the application processes.

Acknowledgements

We would like to thank Werner Nagel and Renate Dohmen from the compute center of the Max Planck Society in Garching (RZG) for their support in experimenting with GENE.

References

1. Geimer, M., Wolf, F., Wylie, B.J.N., Mohr, B.: Scalable parallel trace-based performance analysis. In: Proceedings of the 13th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2006), Bonn, Germany, pp. 303–312 (2006)
2. Gerndt, M., Fürlinger, K.: Specification and detection of performance problems with ASL. Concurrency and Computation: Practice & Experience 19(11), 1451–1464 (2007)
3. Gerndt, M., Kereku, E.: Search strategies for automatic performance analysis tools. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 129–138. Springer, Heidelberg (2007)
4. Miller, B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., Karavanic, K.L., Kunchithapadam, K., Newhall, T.: The Paradyn parallel performance measurement tool. IEEE Computer 28(11), 37–46 (1995)
5. Roth, P.C., Miller, B.P.: The distributed performance consultant and the sub-graph folding algorithm: On-line automated performance diagnosis on thousands of processes. In: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP'06 (March 2006)
6. Shende, S.S., Malony, A.D.: The TAU parallel performance system. International Journal of High Performance Computing Applications, ACTS Collection Special Issue (2005)
7. Wolf, F., Mohr, B.: Automatic performance analysis of hybrid MPI/OpenMP applications. In: Proceedings of the 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2003), pp. 13–22. IEEE Computer Society, Los Alamitos (February 2003)
8. Rivera, O.: Space Filling Curves in Static Load Balance for Seismological Applications. Presentation, Leibniz Rechenzentrum (April 30, 2009)

Performance Simulation of Non-blocking Communication in Message-Passing Applications

David Böhme[1,2], Marc-André Hermanns[1], Markus Geimer[1], and Felix Wolf[1,2]

[1] Jülich Supercomputing Centre
Forschungszentrum Jülich, Germany
{d.boehme,m.geimer,m.a.hermanns,f.wolf}@fz-juelich.de
[2] Aachen Institute for Advanced Study in Computational Engineering Science
RWTH Aachen University, Germany

Abstract. In our previous work [1], we introduced performance simulation as an instrument to verify hypotheses on causality between locally and spatially distant performance phenomena without altering the application itself. This is accomplished by modifying MPI event traces and using them to simulate hypothetical message-passing behavior. Here, we present enhancements to our approach, which was previously restricted to blocking communication, that now allow us to correctly simulate MPI non-blocking communication. We enhanced the underlying trace data format to record communication requests, and extended the simulator to even retain the inherently non-deterministic behavior of operations such as `MPI_Waitany`.

1 Introduction

As a prerequisite for the productive use of state-of-the-art supercomputers, the HPC community needs powerful and scalable performance-diagnosis tools that make the optimization of parallel applications both more effective and more efficient. One major difficulty application developers are confronting with traditional performance tools is that the tools often diagnose only the symptoms of performance problems but not necessarily their causes. Often, the symptoms appear much later or on a different processor than the event causing it. The temporal or spatial distance between cause and symptom constitutes a substantial challenge in deriving helpful conclusions from a set of performance data.

In our earlier work [1], we have presented a simulator called SILAS (SImulation of LArge-Scale parallel applications) that can be used to verify hypotheses on causal connections between different performance phenomena at very large scales. The verification is accomplished by modifying event traces according to a hypothesis and using them to simulate the hypothetical message-passing behavior. The predicted behavior can then be scanned for wait states to investigate how the modification would influence (and hopefully reduce) their occurrence in various parts of the program. Typical questions the simulation can answer encompass how the performance behavior changes if a specific computation is more evenly distributed across the machine or if a specific communication operation is replaced or eliminated. The simulator performs a parallel real-time reenactment of the communication to be simulated using the original execution configuration. This eliminates the need for *modeling* communication and, thus, circumvents a major source of prediction inaccuracy.

So far, our simulator was able to replay only MPI blocking point-to-point and collective communication, but not non-blocking communication, as information on communication requests was not yet recorded in the trace data. In this paper, we outline extensions to the trace format and the simulator itself that allow us to correctly simulate all aspects of MPI non-blocking communication, thus making our simulation approach applicable to a much broader range of MPI applications. Special emphasis is given on the feature of retaining the inherent non-determinism exhibited by operations such as `MPI_Waitany` or `MPI_Test`, which may yield different results during simulation than they did during trace recording.

After discussing related work and briefly recapitulating the working principle of the simulator in the remainder of this section, we describe the required extensions to the trace format in Section 2. In Section 3, we present the basic approach for simulating non-blocking communication and mechanisms to retain non-deterministic behavior in the simulation. Finally, an experimental evaluation to demonstrate the scalability and accuracy of our approach is given in Section 4, before concluding in Section 5.

1.1 Related Work

The principle of trace-driven performance prediction has already been intensively studied. An early performance-analysis toolkit offering trace-based simulation capabilities as one element of a comprehensive feature catalog is AIMS [2], which estimates the scalability of parallel applications by extrapolating previously generated execution traces to higher numbers of processors and larger problem sizes. DIMEMAS [3] provides the ability to simulate the execution behavior of parallel programs based on previously generated event traces. The underlying prediction model allows the adjustment of relative processor speeds, network bandwidth and latency within and across nodes, the number of input and output links, and the processor scheduling policy. Predicting application performance for emerging architectures larger than those at one's disposal is the focus of BigSim [4]. BigSim combines an emulator that is capable of running larger numbers of virtual processes on a smaller number of physical processors with a post-mortem simulator that uses traces generated during an emulated run.

Compared to the approaches described above, our work clearly concentrates on the effects of fine-grained alterations of application-level behavior with respect to the performance under an identical execution configuration. The most important methodological difference is the use of a parallel real-time replay of the simulated communication at the original scale, which offers scalability advantages and relieves us of the burden of modeling the extremely complex communication infrastructures found on today's large-scale machines.

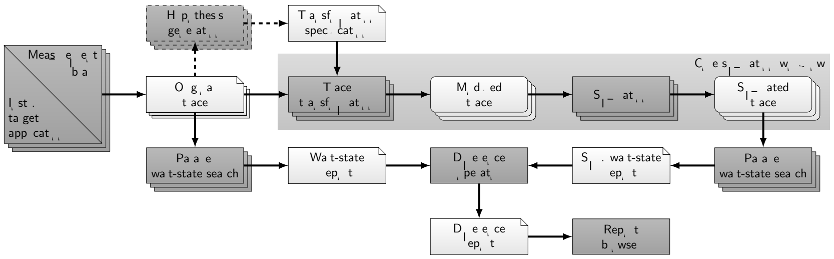


Fig. 1. Workflow for verifying optimization hypotheses. Dark rectangles denote programs, light rectangles with the upper right corner turned down denote files, and light rectangles with rounded corners denote data objects residing in memory. Stacked symbols indicate multiple instances of programs, files, or data objects running or being processed in parallel. The target application generating the event trace is the entry stage of the workflow. Judging the difference between normal execution and the predicted outcome of the optimization displayed in the report browser is the final stage.

1.2 Hypothesis Verification

Here, we briefly review the intended usage scenario for our simulator in the context of the Scalasca toolset [5]. Figure 1 illustrates the role of the simulator in the procedure of verifying hypotheses on causality between temporally or spatially distant performance phenomena. The general objective of the process is to generate wait-state analyses from both the measured and the predicted behavior and compare the results to allow conclusions on the effects of hypothetical program modifications with respect to wait states and other performance metrics. The workflow starts with running the instrumented target application in the execution configuration we want to make predictions for and generating an event trace consisting of one trace file per application process. During all subsequent steps, access to the event trace occurs through a parallel object-oriented high-level API [6]. The primary usage model of the API assumes a one-to-one mapping between application and tool processes, that is, for every process of the target application, one tool process is created which loads the corresponding trace data into main memory and offers random access to individual events. Data exchange among tool processes is accomplished via MPI communication.

A hypothesis includes the specification of a trace transformation, which may prescribe the adjustment of event timestamps, the deletion of existing events, or the insertion of new events to model changes in the application's source code. Currently, a set of parametrized standard transformations including the scaling of functions or the elimination of messages can be specified. After the transformation has been applied, the simulator performs a parallel real-time replay of the events stored in the trace. Computation intervals are simulated simply by elapsing the time in between using busy wait, whereas communications are simulated by reenacting the communication operations recorded in the trace. Thus, the time of a communication is determined by the time needed to execute the corresponding MPI call under modified conditions. As the simulation progresses, event timestamps are adjusted to reflect the time elapsed since simulation start.

2 Trace Format Extensions

The trace format used by our performance analysis and simulation tools stores data in event records. There is a number of fixed event record types, for example for entering or exiting source code regions, or sending and receiving messages, respectively. Each event record contains a timestamp and, according to its type, other data such as the receiving location for send events or a source code region identifier for region-enter events. Event records are written consecutively in the order of the timestamps, with each process writing its own trace file.

In its previous form, our trace format did not provide explicit support for MPI non-blocking communication. Only send start events for `MPI_Isend` and receive completion events for `MPI_Wait*` or `MPI_Test*` regions were recorded using generic send/receive records. While this is sufficient to detect some communication inefficiencies (e.g., Late Sender) in our parallel performance analyzer, it does not allow accurate replay of communication as it is required for the simulation. In particular, neither information on the send completions and receive starts associated with the respective non-blocking send starts and receive completions, nor on failed tests for completion in `MPI_Test` is available in the trace.

2.1 Attribute Records

In order to enhance application traces with additional information at minimal impact on our current code base, we introduced the notion of *attribute records* to store additional, optional data for events. An event can have an arbitrary number of attributes, which are written as attribute records immediately before the corresponding event record in the trace. Unlike event records, attribute records do not contain a timestamp field, which keeps the record size as small as possible. Compared to the alternative approach of adding more fixed-size special-purpose event records, using attribute records to augment a particular event with additional information offers far more flexibility and better extensibility.

2.2 Non-blocking Event Record Types

For a full representation of non-blocking communication, we introduced *request IDs* to identify individual communication requests and to associate a request start with its completion. During trace recording, the opaque `MPI_Request` objects are mapped onto unique request IDs, which are stored in the trace for every non-blocking communication request start and completion.

While we added new event record types to store the request ID for receive starts and send completions, we continue to use the generic point-to-point send and receive record types for send start and receive completion events. Here, the request ID is stored in an attribute to the generic event. Using an attribute

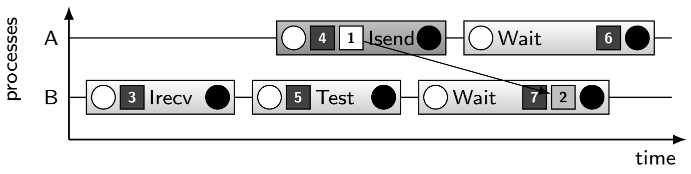


Fig. 2. Trace format extensions. Previous format: (1) send event; (2) receive event. Extensions: (3) receive start; (4), (7) request attribute; (5) test event; (6) send completion. White circles denote region enter events, black circles denote region exit events.

instead of new special-purpose event records keeps backward compatibility and allows us to reuse large portions of existing code in our analysis tools.

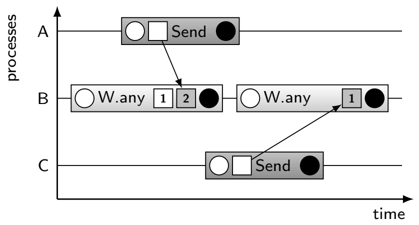
In addition to events for send completion and receive request, we also added a *tested* event, which indicates that a request has been unsuccessfully probed for completion in a call to `MPI_Test` or `MPI_Waitany/some`, and a *cancel* event which indicates a request that has been canceled using `MPI_Cancel`. Figure 2 shows the use of the new event records.

3 Simulation of Non-blocking Communication

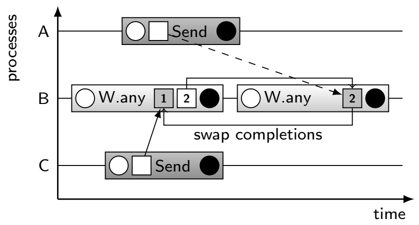
Given a trace enhanced with additional data as described in Section 2.2, replay of deterministic non-blocking communication in the performance simulator is now straightforward. When a non-blocking request start operation is encountered during trace replay, a corresponding `MPI_Isend` or `MPI_Irecv` operation is invoked, and the `MPI_Request` object obtained from MPI is saved in a (request ID, `MPI_Request`) map. Upon request completion, the requests corresponding to the request IDs found in the trace are completed using `MPI_Wait` or `MPI_Waitall`.

3.1 Retaining Non-deterministic Behavior in the Simulation

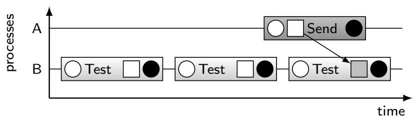
Our basic non-blocking communication simulation approach sketched above allows accurate simulation of non-blocking communication if only `MPI_Wait` or `MPI_Waitall` are used for request completion, but some difficulties arise for inherently non-deterministic operations like `MPI_Test` or `MPI_Waitany/some`. For example, a call to `MPI_Waitany` may yield a different result in the altered, simulated scenario than it did during trace acquisition. Likewise, a test for completion using `MPI_Test` which failed in the original run could succeed in the replay, or vice versa. Essentially, in an application scenario modified according to a performance hypothesis, the order and source code location of request completions can change compared to the original application's behavior, which may have a significant effect on the observed performance characteristics. Restricting request processing in the simulation to the order and locations found in the trace would therefore not accurately predict the application's communication behavior for non-deterministic operations. Hence, our non-blocking communication model needs to take reordering and relocating of request completions into account.



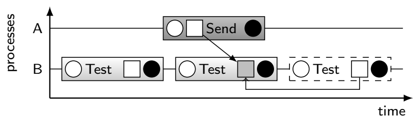
(a) Waitany in original trace: request 2 completes first



(b) In simulation, former request 1 completes first: Completion events are swapped, request ID 1 is remapped



(c) Test in original trace: request completes in second Test



(d) In simulation, test completes earlier. Completion is pre-drawn and additional test call removed.

Fig. 3. Retaining non-deterministic behavior of Waitany and Test

Simulating Waitany. For `MPI_Waitany` regions, the trace contains a completion event record with the request ID that completed in the original run, and *tested* event records with the request IDs that were also passed to the original call to `MPI_Waitany`. In the simulation, the request objects for all given request IDs are passed to `MPI_Waitany`. If the request that completed in the simulation is not the same as in the original run, we swap the completion events and *remap* the request, that is, the remaining events with the request ID that completed in the simulated run are mapped to the ID of the request that completed in the original run (Figure 3a and 3b). As a result, these test or completion events will now be handled for the request that completed originally. Since the positions of subsequent events in the trace pertaining to a certain request ID are known from a preprocessing step, the extra effort for request remapping is negligible. By allowing the simulation in `MPI_Waitany` to complete a request different from the one completed in the original run, we can accurately model the application's intended communication behavior ("return the first request which completes").

Simulating Tests. For calls to `MPI_Test` that are unsuccessful (i.e., do not complete a request), a *tested* event with the corresponding request ID is stored in the trace (Figure 3c). In this case, the simulator calls `MPI_Test` with the associated request. If the request does complete in the simulation, the completion is *pre-drawn*: the test event will be replaced with the requests' completion event, and all remaining test events with this request ID are deleted from the trace. If the last test or completion event remaining in a region is deleted, that region will be removed from the simulation altogether (Figure 3d).

More difficulties arise for MPI_Test calls which are successful in the original run, but fail in the simulation. In this case, the application would try to complete the request again later on. However, the simulation is bound to the trace that was recorded in the original run, which does not contain any information on how the application would have handled the request. Since there is no useful strategy for the simulator to process the request later if the MPI_Test call was unsuccessful, we explicitly complete a request using MPI_Wait if an MPI test operation succeeded in the original application run. This approach may, however, introduce some waiting time which would not have occurred in the modified application.

3.2 Limitations

While our simulator handles non-blocking communication well for most cases, there are a few noteworthy restrictions.

Non-deterministic operations pose a fundamental limitation on our simulator. Applications can take entirely arbitrary actions depending on the outcome of a non-deterministic operation, whereas our simulator is bound to the trace recorded in the original run. In some cases, our heuristics for retaining non-determinism by reordering and relocating request completions may fail to reproduce the application's behavior, or in extreme cases even deadlock. The user can therefore enable a deterministic simulation mode, which restricts request processing to the exact order found in the trace, at the cost of losing some simulation precision. It should be noted, though, that severe problems occur only for pathological cases exhibiting a highly unusual communication behavior. Typical communication patterns, such as looping MPI_Waitany on a fixed list of requests, work as expected. Genuinely reproducing the application behavior for different outcomes of MPI_Test operations is not possible without knowledge of the application semantics, which currently exceeds the scope of our replay approach. As such, our heuristic represents a best-effort approach which at least allows conclusions, e.g., on the number of tests needed to complete a request.

Also, our model currently does not handle persistent communication requests explicitly. Instead, they are handled as ordinary non-blocking communication, which may slightly overestimate the processing time for those requests in the simulation.

4 Results

We conducted a number of experiments to demonstrate accuracy and scalability of our approach, using small synthetic benchmarks and more complex real-world benchmark codes. All experiments were performed on the 72-rack Blue Gene/P supercomputer Jugene and the 448-core Power6 cluster Jump at the Jülich Supercomputing Centre.

4.1 Simulation Accuracy

One effective way of validating the simulation accuracy is an *identity simulation*, where a simulation run without any performance hypothesis applied is compared to the original program behavior. We conducted identity simulation experiments with bt from the NAS parallel benchmark suite [7] with 256 and 1024 processes on Jugene. The runtime of the benchmark kernel in the original, uninstrumented benchmark executable is compared with the runtime during trace recording and the simulated runtime. The results are shown in the following table.

Table 1. NAS bt measurement and simulation results

	Comm.	**Runtime (sec)**			**Deviation from Original**	
No. procs	fraction	Original	Traced	Simulated	Traced	Simulation
256	16 %	46.56	47.23	46.82	1.44 %	0.56 %
1024	30 %	14.93	16.57	15.67	10.98 %	4.96 %

Note that the deviation between original and traced runtime is about 2.5 times (256 procs) and 2.2 times (1024 procs, respectively) higher than the deviation between original and simulated runtime. Synthetic experiments confirm that the overhead of tracing is indeed higher than the overhead created by the replay of communication in the simulator. Especially small, short-running functions like `MPI_Irecv` can have a high relative tracing overhead.

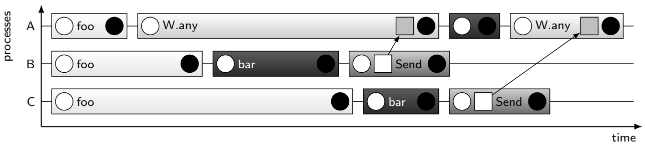
In general, inaccuracies introduced by tracing may also negatively influence the simulation, which is based on the trace. While the overall simulation accuracy for non-blocking communication in the presented case is good, the relation between simulation accuracy and measurement overhead during trace recording still requires further investigation.

4.2 Non-deterministic Behavior Simulation

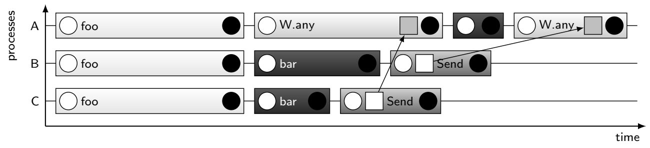
A simple synthetic benchmark demonstrates the necessity of retaining non-deterministic behavior in the simulation. Figure 4a outlines the working principle. The master process is waiting for messages from the remaining processes in a loop using `MPI_Waitany`. Due to a load imbalance in code region `foo`, the messages arrive in the order of process ranks.

We recorded an example trace of the program with four processes on our Power6 cluster and performed simulation runs both with request relocation and reordering enabled (non-deterministic mode) and disabled (deterministic mode). First, we performed an identity simulation, then another simulation with a performance hypothesis to balance code region `foo` applied. The result of the latter was compared to a modified version of the original program with `foo` balanced.

Table 2 shows the results of the experiments. While for the identity simulation, both deterministic and non-deterministic mode yield accurate results, only the non-deterministic mode is able to predict the program's behavior with region `foo`



(a) Load imbalance in `foo`: messages to arrive in order of ranks



(b) Balancing `foo`: message order is reversed on destination

Fig. 4. Waitany Benchmark: Original (a) and modified version (b)

Table 2. Non-deterministic behavior simulation: Deviation of simulation result from original (identity simulation) and modified program behavior (balance experiments)

	Runtime		**Identity simul. Δ**		**Balanced simul. Δ**	
Metric	original	modified	non-det.	det.	non-det.	det.
Total time	34.82 s	31.65 s	0.0003 %	0.0003 %	0.0689 %	8.14 %
Point-to-Point	5.60 s	2.40 s	0.0013 %	0.0009 %	0.0029 %	33.34 %
Late Sender	5.59 s	2.39 s	0.0041 %	0.0036 %	0.0083 %	33.36 %
Synchronization	2.70 s	2.70 s	0.0048 %	0.0011 %	0.0022 %	66.67 %

balanced correctly. This is because by balancing region `foo`, the order of message arrival in the `MPI_Waitany` call is reversed due to another load imbalance in code region `bar` (Figure 4b). By retaining non-deterministic behavior in our simulator, we can predict this effect correctly. In deterministic mode, however, the simulator is restricted to the original order of message arrival in the program, and therefore introduces larger waiting times.

5 Conclusion

We have presented enhancements to our performance simulator and underlying trace data format which allow us to accurately simulate the message-passing behavior of applications that utilize MPI non-blocking communication. Using both new event records and attribute records, we could amend application traces with communication request tracking capabilities requiring only minimal changes to the existing code base of our analysis tools. Moreover, attribute records may provide a generic and flexible approach to enhance application traces with additional, optional information. By reordering and relocating communication requests, our simulator can accurately predict even non-deterministic communication behavior for most typical communication patterns.

Further enhancements we plan to incorporate into our simulator are explicit support for persistent communication requests and support for MPI-2 one-sided communication. We are also investigating more detailed performance analysis procedures for non-blocking communication using the new request tracking capabilities in our parallel performance analyzer.

Acknowledgment

Financial support from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111 and from the Helmholtz Association of German Research Centers under Grant No. VH-NG-118 is gratefully acknowledged.

References

1. Hermanns, M.A., Geimer, M., Wolf, F., Wylie, B.J.N.: Verifying causality between distant performance phenomena in large-scale MPI applications. In: Proceedings of the 17th International Conference on Parallel, Distributed, and Network-Based Processing (February 2009)
2. Yan, J., Sarukkai, S., Mehra, P.: Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs using the AIMS Toolkit. Software – Practice and Experience 25(4), 429–461 (1995)
3. Rodriguez, G., Badia, R.M., Labarta, J.: Generation of simple analytical models for message passing applications. In: Danelutto, M., Vanneschi, M., Laforenza, D. (eds.) Euro-Par 2004. LNCS, vol. 3149, pp. 183–188. Springer, Heidelberg (2004)
4. Zheng, G., Wilmarth, T., Jagadishprasad, P., Kalé, L.V.: Simulation-based performance prediction for large parallel machines. International Journal of Parallel Programming 33(2-3) (2005)
5. Geimer, M., Wolf, F., Wylie, B.J.N., Mohr, B.: Scalable parallel trace-based performance analysis. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 303–312. Springer, Heidelberg (2006)
6. Geimer, M., Wolf, F., Knüpfer, A., Mohr, B., Wylie, B.J.N.: A parallel trace-data interface for scalable performance analysis. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, pp. 398–408. Springer,
Heidelberg (2007)
7. Bailey, D.H., Barzcz, E., Dagum, L., Simon, H.D.: NAS parallel benchmark results. IEEE Parallel Distrib. Technol. 1(1), 43–51 (1993)

Capturing and Visualizing Event Flow Graphs of MPI Applications*

Karl Fürlinger[1] and David Skinner[2]

[1] Computer Science Division, EECS Department
University of California at Berkeley
Soda Hall 593, Berkeley CA 94720, U.S.A.
fuerling@eecs.berkeley.edu
[2] Lawrence Berkeley National Lab
Berkeley, California
deskinner@lbl.gov

Abstract. A high-level understanding of how an application executes and which performance characteristics it exhibits is essential in many areas of high performance computing, such as application optimization, hardware development, and system procurement.

Tools are needed to help users in uncovering the application characteristics, but current approaches are unsuitable to help develop a structured understanding of program execution akin to flow charts. Profiling tools are efficient in terms of overheads but their way of recording performance data discards temporal information. Tracing preserves all the temporal information but distilling the essential high level structures, such as initialization and iteration phases can be challenging and cumbersome.

We present a technique that extends an existing profiling tool to capture event flow graphs of MPI applications. Event flow graphs try to strike a balance between the abundance of data contained in full traces and the concise information profiling tools can deliver with low overheads.

We describe our technique for efficiently gathering an event flow graph for each process of an MPI application and for combining these graphs into a single application-level flow graph. We explore ways to reduce the complexity of the graphs by collapsing nodes in a step-by-step fashion and present techniques to explore flow graphs interactively.

1 Introduction

Understanding performance characteristics of applications at a high level is essential in many diverse areas of high performance computing. Application developers, hardware engineers, or computing center support and procurement experts use tools to establish that the application uses the available resources efficiently or if there is potential for improvement.

* This work was supported by the National Science Foundation under award number 0721397. Additional support was provided by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227).

Many techniques made available by current tools are insufficient for getting a high-level understanding of the "execution flow" of an application. Most performance tools can be categorized into either profiling or tracing. Profiling tools are efficient in terms of overheads but their way of recording performance data discards temporal information. Tracing preserves all the temporal information but uncovering the essential high level structures, such as initialization and iteration phases can be challenging and cumbersome.

We present a technique that extends an existing profiling tool to capture event flow graphs of MPI applications with very low overhead. Event flow graphs try to strike a balance between the abundance of data contained in full traces and the concise information profiling tools can deliver with low overheads. The graphs are similar in concept to flow charts used to describe algorithms and design software systems.

We describe our technique for efficiently gathering an event flow graph for each process of an MPI application and for combining multiple graphs into a single application-level flow graph. We explore ways to reduce the complexity of the graphs by collapsing nodes in a step-by-step fashion and present techniques to explore flow graphs interactively.

The rest of this paper is organized as follows: In Sect. 2 we give a short overview of the integrated performance monitoring (IPM) tool that we extended to capture event flow graphs. In Sect. 3 we describe our approach to recording the flow graphs in MPI applications, and in Sect. 4 we describe techniques for the interactive visualization and exploration of the graphs and apply the tool to some example applications. In Sect. 5 we survey related work and in Sect. 6 we conclude and discuss areas for future work.

2 Application Profiling and Workload Characterization with IPM

IPM is a profiling and workload characterization tool for MPI applications. IPM achieves its goal of minimizing monitoring overhead by recording performance data in a fixed-size hash table resident in memory and carefully optimizing time-critical operations. At the same time, IPM offers very detailed and user-centric performance metrics. IPM's performance data is delivered as an XML file that can subsequently be used to generate HTML pages, avoiding the need for special graphical user interfaces. Pairwise communication volume between processes, communication time breakdown across ranks, MPI operation timings, and MPI message sizes (buffer lengths) are some of IPM's most widely used features. IPM is available from `http://ipm-hpc.sourceforge.net` for download and is distributed under the LGPL license.

3 Recording Event Flow Graphs of MPI Applications

We assume the following general model of performance monitoring for MPI applications: An MPI application is composed of n *processes* each identified by an integer in $[0, \ldots, n-1]$, its *rank*. A set of events $E_i \subseteq E$ happen in each process i. We do not further formally specify what the events are, but we assume they occur at a certain time and have duration. Each event e has an associated *signature* $\sigma(e) \in S$ which captures the characteristics we are interested in. $\sigma : E \mapsto S$ is the signature function. Concretely we think of a signature $\sigma(e)$ as a k-tuple $\sigma(e) = (\sigma^1(e), \sigma^2(e), \ldots, \sigma^k(e))$, where each $\sigma^j()$ is a signature *component*. Useful components of signature functions are listed in Fig. 1.

Signature component	**Signature function**	**Data type**	**Typical Size (#bits)**
Wallclock time	$time(e)$	floating point	32/64
Sequence number	$seq(e)$	integer	32
Type of MPI call	$call(e)$	integer	8
Data size	$size(e)$	integer	32
Data address	$address(e)$	integer	64
Own rank	$rank(e)$	integer	32
Partner rank	$partner(e)$	integer	32
Callsite ID	$csite(e)$	integer	16
Program region	$region(e)$	integer	8

Fig. 1. Components of an event signature function

Our goal for performance observation is to get an event inventory of an application (i.e., understand the events that happened and their characteristics) by associating performance data (number of occurrences, statistics on the duration) with event signatures. If the signature includes $time()$ this essentially models tracing; if it does not we have a model for profiling.

IPM is a profiling tool and for efficiency reasons we would like to keep the signature space much smaller than the event space ($|E| >> |S|$). In this case the signature function is not injective and performance data can be envisioned as a table indexed by the signature, with a number of columns for the statistics we are interested in. In IPM we implement this indexing using a hash table resident in memory, the hash keys are 64 to 128 bits long and the hash values are on the order of 160 bits (20 bytes) big.

Evidently, if the signature does not include $time()$ or $seq()$ we lose the temporal dimension of the performance data, and with it the ability to understand which events happened before or after each other from the measured data. In this paper we show that some important temporal information can be recovered by keeping track of the sequence of event signatures. We call the resulting graphs which are akin to control flow graphs event signature flow graphs or simply event flow graphs.

To construct a flow graph consider an application executing with n processes and let $E_i = \{e_0, e_1, \ldots\}$ be the sequence of events at rank i, $\sigma : E_i \mapsto S_i$ be the signature function at rank i, and $s_i^0 \in S_i$ some initial signature value. Then σ' with

$$\sigma'(e_0) = (s_i^0, \sigma(e_0))$$

$$\sigma'(e_i) = (\sigma(e_{i-1}), \sigma(e_i)) \qquad (i > 0)$$

is the history signature for σ. The directed weighted graph $G = (N_i, L_i, w_i, s_i^0)$ with

$$\begin{aligned} N_i &= \{\sigma(e_i)\} \qquad e_i \in E_i \\ L_i &= \{\sigma'(e_i)\} \qquad e_i \in E_i \\ w_i : L_i \mapsto \mathbb{N} \qquad w_i(l) &= |\{e_i : \sigma'(e_i) = l\}| \qquad l \in L_i \end{aligned}$$

is the event signature flow graph for rank i and s_i^0 is the start node of the graph. An example flow graph is shown in Fig. 3. Nodes correspond to MPI calls, edges between the nodes correspond to transitions between them.

3.1 Merging Graphs from Multiple Processes

Event flow graphs form different processes can be merged in a straightforward way to form a multigraph (a graph with multiple edges between a pair of nodes). We build the merged graph by identifying similar nodes among the graphs (i.e., having identical $\sigma(e_i)$ with respect to some equality criterion) and inserting the edges and weights from each depending on the signature component we are concerned with.

The best way for identifying two event signatures $\sigma(e_i)$ as being identical depends on the signature components:

Signature component	**Equality test**
Type of MPI call	Exact equality
Data size	Exact equality or approximate (same magnitude) equality
Data address	Exact equality
Own rank	Discarded, since we merge across ranks
Partner rank	Equality of relative ranks
Callsite ID	Equality in unified calltrees
Program region	Exact equality

Fig. 2. Unification of signature components across MPI processes

Identifying identical callsite IDs requires us to unify the calltree of each rank. During the execution a calltree is recorded (nodes are the callsites of MPI calls) and numerical IDs are assigned consecutively. Depending on the sequence in which functions are executed, the same callsites can be assigned different IDs on different processes. A unification step which IPM performs after the program terminates guarantees a consistent assignment of callsite IDs.

For comparing ranks we use differences (relative ranks) since in many applications parallelism is exploited in the form 2-D or 3-D domain decomposition and the MPI communication pattern is often based on the topological position of a processor (i.e, nearest neighbor communication in a grid). For this reason it is most often convenient to convert the absolute partner rank of a communication event into a relative rank (e.g., a `MPI_Send` to processor with relative rank -4).

One further simplification step can be performed on the edges between nodes. To simplify presentation and understanding of the graphs we cluster edges with the same multiplicity or weight together.

An example for a resulting application level event flow graph for a very simple application is shown in Fig. 3. The program is executed with four MPI processes, where ranks $0, 2$ perform a receive operation and ranks $1, 3$ perform a send.

```
void main(int argc, char* argv[]) {
  MPI_Init(...);
  MPI_Comm_size(...);
  MPI_Comm_rank(..., &myrank);

  for(i=0; i<10; i++) {
    if(myrank is odd)
      MPI_Send(10 doubles to rank -1);
    else
      MPI_Recv(10 doubles from rank +1);
  }
  MPI_Finalize();
}
```

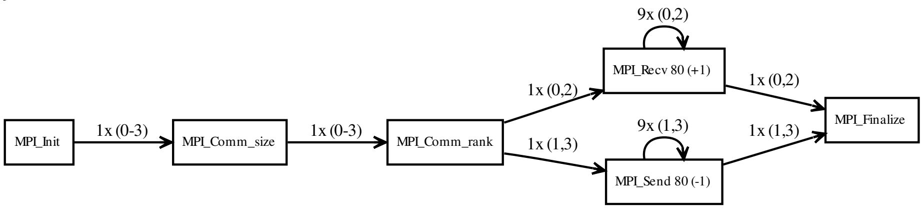


Fig. 3. A simple MPI program, executed with four MPI processes and its accompanying merged event flow graph using relative rank addressing

3.2 Implementation in IPM

We have implemented the event flow recording scheme as described in Sect. 3 in our profiling tool IPM. IPM keeps event statistics (number of occurrences, total duration, and so on) in a hash table and the hash key derived from the MPI communication events correspond to the event signatures. To record the event flow information, the hash key is extended to contain both the current signature $\sigma(e_i)$ as well as the signature of the previous event $\sigma(e_{i-1})$. The previous event's signature is recorded in a variable and updated on each insert into the hash table.

Using this scheme event statistics are now correlated with pairs of event signatures that form the edges of the event flow graph. Upon program termination, the hash table is inspected and the flow graph is reconstructed from the hash table by looking for matching pairs of event flow edges.

4 Visualizing and Exploring Event Flow Graphs

The flow-graphs are recorded by IPM on a per-rank basis and written to a file. The merging and unification step is performed by a perl script in a post-processing step which generates a number of event flow graph files suitable for input into Graph::Easy [1] and further layout by dot [3].

Consider the table in Fig. 4. It shows the number of events in a full trace of several applications of the NAS parallel benchmark suite as well as the number of nodes in the event flow graph using the signature components indicated in the first column. These application contain no developer-provided phase markers and the MPI call type can be derived from the callsite ID, so *size*(), *partner*(), *csite*() provide the largest signature space and a subset of these signature functions will generally lead to fewer nodes in the flow graphs.

Evidently, the callsite ID is essential to achieve a large signature space. In fact, adding *partner*() and *size*() components does not add more nodes to the flow graphs for all but one application (MG). For MG both the communication partner and the transmit data size need to be added to differentiate between all events.

Fig. 5 shows the flow graph of the IS application. For this small application the entire flow graph is easily visualized. For larger applications the direct approach becomes infeasible. Considering the results from Fig. 4, we decided to focus the on methods to interactively explore the event flow graphs along the callsite dimension by developing a combined calltree-eventgraph display.

The user is presented with a calltree display alongside with a *portion* of the flowgraph which depends on the node selected in the calltree. An example for this is shown in Fig. 6. The leafes of the calltree on the left correspond to the MPI events that comprise the flowgraph on the right.

Assume a user selects an internal node `foo()` of the calltree. Then there is effectively a partitioning of the flowgraph nodes into three sets: (1) nodes that are immediate children of the selected node, (2) those that are children but not immediate children, and (3) all other nodes.

Method	**BT**	**CG**	**EP**	**FT**	**IS**	**LU**	**MG**	**SP**
Full Trace	29856	20184	36	85	165	255213	8988	49828
Event Flow Graph:								
size(), *partner*(), *csite*()	404	240	36	45	57	277	2796	352
size(), *partner*()	184	72	28	36	45	93	236	124
size(), *csite*()	404	240	36	45	57	277	2644	352
partner(), *csite*()	404	240	36	45	57	277	1140	352
size()	76	60	28	36	45	75	220	76
partner()	76	48	24	32	41	56	60	60
csite()	404	240	36	45	57	277	852	352

Fig. 4. Number of events in the full traces and number of nodes in the event flow graphs for the NAS parallel benchmark suite (size A, 4 processors)

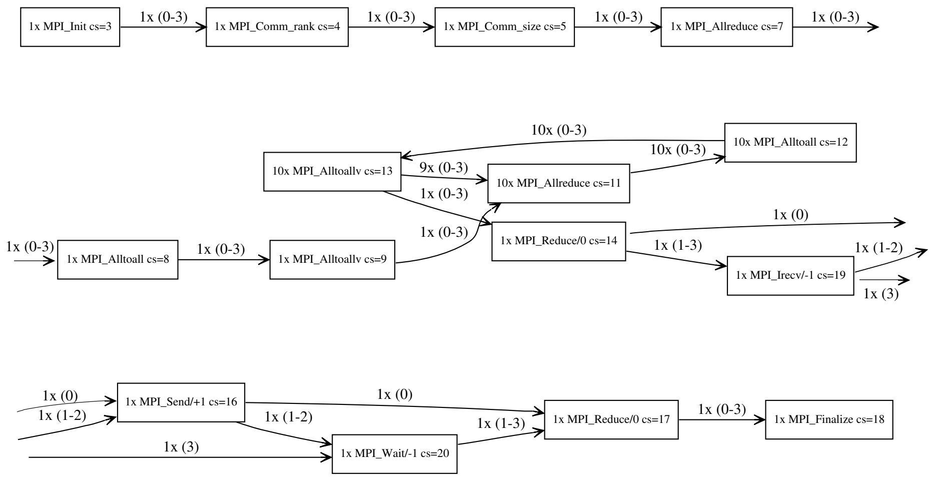


Fig. 5. Full event flow graph of the IS application from the NAS parallel benchmark suite

Set (1) corresponds to MPI functions called directly from `foo()` (i.e, leaves one level below `foo()`; these nodes and their transitions are shown directly in the flowgraph display to the right. Nodes in set (2) correspond to functions called from functions called from `foo()` (leaves two or more levels below `foo()`). Those nodes are replaced by a representative, which is the child function called from `foo()` that leads to their execution. Nodes in set (3) are not displayed at all unless there is a transition to a visible node (from sets (1) or (2)). In this case the node is displayed with a dotted border and a dotted line, indicating a control flow coming from the "outside".

An example of this display technique is shown in Fig. 6. This method is very effective at narrowing down the set of nodes in the flow graph to a manageable set for interactive exploration and understanding of application code.

5 Related Work

Control flow graphs are an important topic in the area of code analysis, generation, and optimization. In that context, CFGs are usually constructed based on a compiler's intermediate representation (IR) and are defined as directed multi-graphs with nodes being basic blocks (single entry, single exit) and nodes representing branches that a program execution *may* take. The difference to the CFGs in our work is primarily twofold. First, the nodes in our graphs are not basic blocks but communication events. Second, the edges in our graphs record transitions that have actually happened during the execution and also contain a count that shows how often the transition occurred.

Dragon [2] is a performance tool from the OpenUH compiler suite. It can display static as well as dynamic performance data such as the calltree and control flow

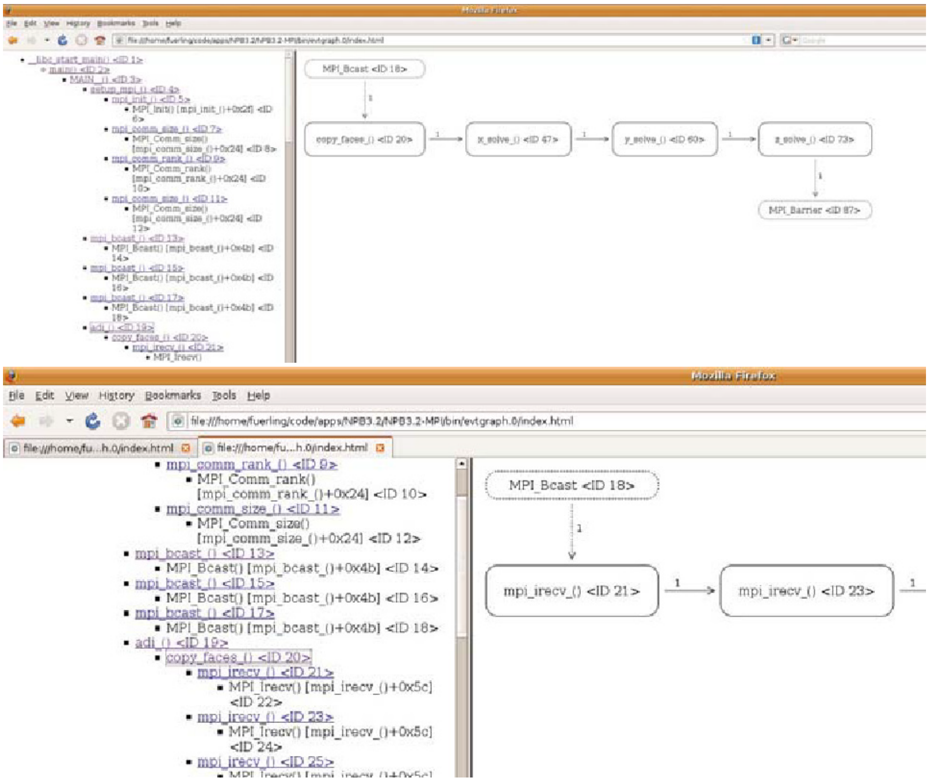


Fig. 6. Combined calltree-controlflow visualization. The user selects a node on the calltree to the left and depending on the selection only a subset of the event flow graph is presented on the right.

graph. The static information is collected from OpenUH's analysis of the source code, while the dynamic information is based on the feedback guided optimization phase of the compiler. In contrast to our approach, the displays are based on the compiler's intermediate representation of source code. The elements of our visualization are the constructs of the user's model of execution to contribute to a high-level understanding of the program execution characteristics.

The work of Preissl et al. [5] tries to detect recurring patterns of communication events for optimization purposes. Events are recorded as an array of 32-bit integer values (i.e., a trace) and repeating sequences of events are searched for by either a convolution or suffix-tree based method. The identified and matched repeating sequences, together with source code analysis using Rose [6], are the basis for source code transformations such as replacing a series of point to point operations with the corresponding collective. Compared to their method, our technique avoids the overhead of generating, storing, and analyzing traces. Instead our technique of recording the execution control flow directly exposes repetitive structures as loops in the flow graphs.

Finally, the work of Noeth [4] shares some similarities with our approach. In this trace compression scheme, region descriptors are applied to perform both intra-node and inter-node compression. Although this approach is able to reduce traces from applications employing regular communication patterns to near constant size independent of the number of nodes, runtime overhead is incurred for establishing and maintaining the region descriptors. In contrast, our approach has negligible cost at runtime, while we don't guarantee that the trace can be recovered completely. In fact, the potential to recover the original trace employing node ordering heuristics is part of our ongoing work.

6 Conclusion

We have discussed a technique to efficiently gather an event flow graph from MPI applications. Nodes in the graph are representations of MPI communication events and edges represent the number of transitions between them. Event flow graphs try to strike a balance between the abundance of data contained in full traces and the concise information profiling tools can deliver with low overheads. The graphs are conceptually similar to flow charts used in algorithm and application development. We presented ideas to reduce the complexity of the graphs by collapsing nodes in a step-by-step fashion and presented techniques to explore flow graphs interactively.

Future work is planned with respect to several directions. First, while timing statistics are already recorded for each edge of the flow graph, they are currently not used in the visual display. It should be straighforward to develop a coloring scheme to color nodes according to MPI communication time and the data volume sent or received. This would draw the user's attention to the most interesting parts of the graph for optimization purposes.

As a bigger step we plan to explore the usability of the flow graphs to perform MPI process clustering at petascale. With very large numbers of MPI processes used at that scale, performance data visualization that involves the rank ID as a dimension becomes impractical or even impossible. An automated clustering of ranks into a small number of processes that qualitatively exhibit the same behavior would be a solution to this problem. Another area for future exploration is the application of techniques from graph theory to our flow graphs. Examples include cycle detection and extraction to automatically delineate computational and iterative phases.

References

1. The graph:easy web page, `http://search.cpan.org/~tels/Graph-Easy/`
2. Hernandez, O., Liao, C., Chapman, B.: Dragon: A static and dynamic tool for OpenMP. In: Chapman, B.M. (ed.) WOMPAT 2004. LNCS, vol. 3349, pp. 53–66. Springer, Heidelberg (2005)
3. Koutsofios, E., North, S.C.: Drawing graphs with dot. Murray Hill, NJ (October 1993)

4. Noeth, M., Mueller, F., Schulz, M., de Supinski, B.R.: Scalable compression and replay of communication traces in massively parallel e nvironments. In: Proceedings of the 21th International Parallel and Distributed Processing Symposium (IPDPS '07), pp. 1–11. IEEE, Los Alamitos (2007)
5. Preissl, R., Schulz, M., Kranzlmüller, D., Supinski, B.R., Quinlan, D.J.: Using MPI communication patterns to guide source code transformations. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 253–260. Springer, Heidelberg (2008)
6. Quinlan, D.J.: ROSE: Compiler support for object-oriented frameworks. Parallel Processing Letters 10(2/3), 215–226 (2000)

Scalable Event Trace Visualization

Kathryn Mohror[1], Karen L. Karavanic[1,2], and Allan Snavely[2]

[1] Portland State University
[2] San Diego Supercomputer Center
kathryn@cs.pdx.edu, karavan@cs.pdx.edu, allanesnavely@gmail.com

Abstract. Parallel event trace visualizations can aid in discovery of the root causes of certain performance problems on high-end systems. However, traditional trace visualizations are not inherently scalable and require considerable effort on the part of the user to identify similarities and differences in performance across parallel entities. In this work, we evaluate several methods for deciding when traces of different processes in a run are similar enough that only one of the traces needs to be retained and rendered in the visualization. We show visualizations of reduced traces and evaluate them for compression, error, and retention of correct diagnostic information.

1 Introduction

Performance analysts working on today's high-end systems require event-based measurements to correctly identify the root cause of certain performance problems [3]. For example, traces of MPI function entries and exits might be collected to analyze communication overhead. The data might then be analyzed automatically with a tool such as Scalasca [5], or manually, using a trace visualization tool such as Jumpshot [21]. Current trace visualization tools commonly present Gantt charts, showing a bar plot of event occurrences over time, left to right, with one bar per process or task (See Fig. 5.). Generally, the visualization initially shows the entire timeline, and the user has the option to zoom in on portions of the timeline, and possibly on specific ranks, to see more detail.

Traditional event tracing tools often cannot scale up to the demands of current high end applications. The number of events can become quite large, especially for long-running and highly-parallel executions. For example, in one study, we encountered event counts on the order of 10^{10}, for 32-process runs of Sphot that only ran for a few minutes [14]. The file size of the merged trace was 424 GB. At the high end, full-scale trace visualizations quickly exhaust resources such as disk space, and become extremely difficult to read, as the tool user must scroll through thousands of processes and lengthy time lines. It becomes a matter of either being able to see the whole picture, but not being able to see enough detail to draw conclusions about patterns in the trace; or being able to see the needed details, but losing the perspective of the whole picture. The scalability of trace visualizations is not a new topic [7, 8, 9, 13, 16]; however, the continuing upward scaling of high end systems drives a continuing need for more scalable solutions.

We address this goal using a similarity-based inter-process trace reduction technique. We select representative traces from a subset of the processes in an execution and discard the trace data from the remaining processes. Candidate traces for discard are chosen based on similarity to representative traces, evaluated on a pairwise basis. This inter-process reduction has the potential to vastly improve the scalability of traditional trace visualization, making it possible to see more details and more of the whole picture at the same time. Removing similar processes has the effect of reducing the number of bars, increasing the chance for the data to fit on one or two screens, thereby reducing top to bottom scrolling. The goal is to convey important performance-related information about the application such as whether all processes are exhibiting the same behavior, in a way that still works for very high scale runs. An alternative approach, which we have explored in previous work [15], would be to reduce the number of events within each trace, thereby decreasing the left to right length of each bar.

A good trace reduction approach for visualization should lead to a significant reduction in the number of process bars for the common cases, with good scaling behavior and an acceptable amount of processing overhead. Data reduction and overhead are straightforward to evaluate; however, evaluating retention of performance trends is challenging. When we represent a process' behavior using a representative trace from another process, we introduce error. Depending on the algorithm used for deciding if traces match, we might introduce errors in event measurements, message passing parameters, and possibly even event occurrence and ordering. To address this, we present a method for deciding whether these errors change the conclusions that an analyst would reach when looking at the reduced trace, compared to those reached when looking at the full trace.

2 Related Work

Trace Reduction. Aguilera et al. [2], Nickolayev et al. [17], and Lee et al. [12] apply statistical clustering to traces and select a representative trace for each cluster of processes. Both Nickolayev and Lee use the Euclidean distance for clustering, while Aguilera uses a distance metric based on the amount of communication between two processes. Noeth et al. detect patterns of MPI calls and store a single representative of each pattern, and optionally statistical performance measurements [18, 19]. Their focus is on collecting traces for the purpose of simulation, while our focus is on traces for performance analysis and visualization.

Visualization. Freitag et al. reduce traces and data in visualizations by finding intra-process patterns, while Knüpfer et al. reduce by identifying intra- and inter-process patterns. Freitag et al. show a fixed number of repeated patterns for each thread or process [4]. Knüpfer et al. present visualizations in Vampir NG with repeated patterns for each thread or process shown as color blocks that can be interactively decomposed [11]. Nickolayev et al. present an inter-process reduction and show a representative trace for each cluster of processes [17]. Our work most closely resembles that of Nickolayev et al. because we focus solely on inter-process reduction as well. We

```
int main(){
        start_segment("init");
        MPI_Init();
        end_segment("init");
        for(i=0; i < 100; ++i){
            start_segment("main.1");
            do_work();
            MPI_Allgather();
            end_segment("main.1");
        }
        start_segment("final");
        MPI_Finalize();
        end_segment("final");
}
```

Fig. 1. Segment Context Marking. We show a single function, `main()` with the instructions added to mark the segment contexts. We mark initialization, finalization, and all loops. Segment marking is automated using a dynamic instrumentation library.

differ in that Nickolayev et al. implement trace matching for windows of time in the trace and store a representative for each window, while we require the entire trace to match and store the entire representative trace.

3 Our Approach

We explore several methods and criteria for deciding similarity of traces with the goal of reducing the number of representative traces that need to be displayed by a visualization tool. In this section, we present our trace matching methodology, two relaxations on matching criteria designed to increase matches while still retaining the important performance behaviors, and the criteria we use to evaluate the matching methods.

3.1 Trace Matching Methodology

We collected full traces of time stamped function entries and exits for the benchmarks and application as follows. We refer to the trace for a single process simply as a *trace*. We broke the traces into sections we call *segments*. We mark segments as shown in Fig. 1. The *segment context* is the section of code, for example, the main.1 loop in Fig. 1. We used the dynamic instrumentation library Dyninst [10] to instrument the full application for both function entry and exit tracing as well as inserting segment begin and end markers. The simple benchmarks were marked manually.

Given two traces with equal numbers of segments, we compare each pair of segments in order and determine if they are similar. If all segments in both traces are deemed similar, we say that the traces *match* and retain a single *representative trace*. After comparing all traces, we end up with a *set of representative traces* that can be merged for visualization into a *merged representative trace*. We give an example of trace matching in Fig. 2. In addition to comparing event measurements, we also check message passing parameters: source/target rank, bytes transferred, message tags, and

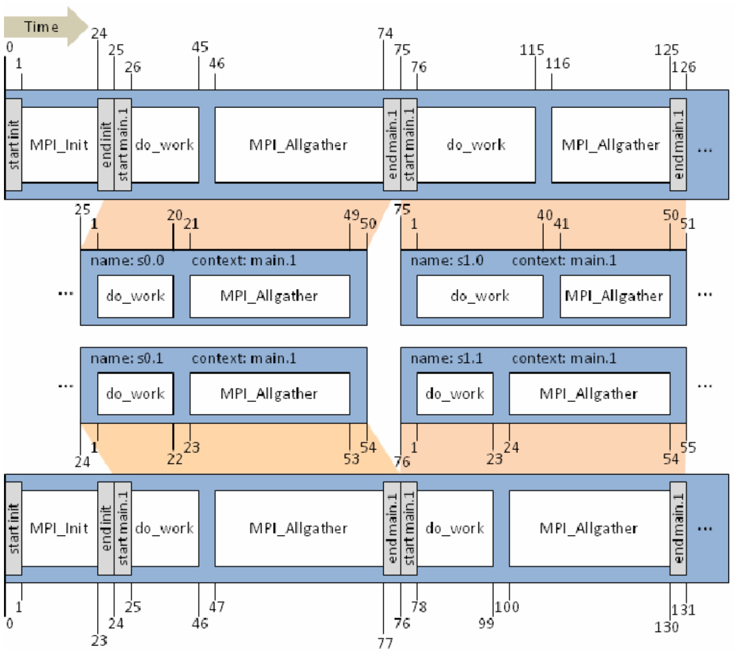


Fig. 2. Segment Matching. The top and bottom bars represent traces for different ranks of the program in Fig. 1. Time values on the bars increase from left to right. Segments markers are gray rectangles with text that tells the segment context. Events are white boxes. Between the traces, we show the result of segmentation. We name the segments *s0.x* and *s1.x*; *x* indicates the rank that wrote the trace. In the segments, the time stamps for the events and segment end times are adjusted relative to the segment start time. To decide matching, we examine the segments pairwise in order, comparing segment start times and all event timings.

communicators. All parameters save the source/target rank must be identical; the source/target rank can be either the same offset, e.g. rank+1 in a nearest neighbor communication pattern, or the same rank, e.g. all ranks send to rank 0.

3.2 Matching Criteria Relaxation

Strictly speaking, segment matching requires that all events be sufficiently similar, including any message passing parameters. However, ignoring some segments or message passing parameters may result in a higher rate of matching without loss of important information. We consider two relaxations:

Ignore Segments: There are cases when we may want to relax the requirement that traces contain all the same events in the same order. For example, in MPI programs, the process with rank 0 is often treated specially and given additional tasks, such as extra function calls in initialization and finalization segments that aren't important to overall performance. If we selectively ignore these segments when matching traces, we may identify more trace matches. In this study, we optionally ignore the initialization and finalization segments.

Relax Messages: By default, we require that all message passing parameters save source/target rank match exactly, meaning we are more closely preserving the behavior across processes. However, two processes could exhibit analogous behavior with different message parameters, and relaxing the requirements could increase matches. We optionally relax checks on message tags and bytes transferred.

In later sections, we refer to matching with no relaxation as *none*, ignoring segments as *ignore*, and relaxing checks on message tags and bytes as *msgs*.

3.3 Evaluation Criteria

We use three criteria to evaluate the methods and relaxation strategies:

Trace Matching. We present percentage file size and degree of matching to show trace reduction. The percentage file size gives a relative measurement of the size of the merged representative traces to the size of the merged original traces. The degree of matching is the ratio of the number of matches to the number of possible matches, which is limited by the structure of the program.

Trace Error. We present the approximation distance metric as the amount of error in a reduced trace as the 90th percentile of absolute error between the measurements in the original and representative traces.

Retention of Performance Trends. Arguably, the most important criterion for evaluating a trace matching metric for performance analysis is deciding whether or not the reduced trace still indicates the same performance problems as the full trace. To evaluate retention of performance trends, we use an automated performance diagnosis tool called Scalasca. [5]. Scalasca parses a merged trace file, producing a hierarchical list of performance diagnoses [20] and their relative severity. We convert the Scalasca visualization into a compact form for our comparative analysis.

4 Experimental Design

In this section, we detail the test programs we used and the methods for deciding trace matching. For our study, we reduced traces of benchmarks with known performance behavior and an application.

Benchmarks. We created benchmarks with performance problems that require tracing for correct diagnosis using the APART Test Suite (ATS), a collection of utilities designed to create programs with known behavior [6]. We chose three example benchmarks to illustrate our approach: dyn_load_balance, early_gather, and late_reciever. The early_gather and late_sender benchmarks exhibit very regular behavior in each iteration. In early_gather, rank 0 is always early to the gather operation. In late_sender, the even ranks are late to the sending operations, causing the odd ranks to block in receives. The dyn_load_balance benchmark has different behavior in each iteration. The upper half of ranks get progressively more work over time until a simulated load balancer is triggered and the work is evenly distributed.

Application. We chose Sweep3D 2.2b, a structured mesh application that computes a 1-group time-independent discrete ordinates three-dimensional Cartesian geometry

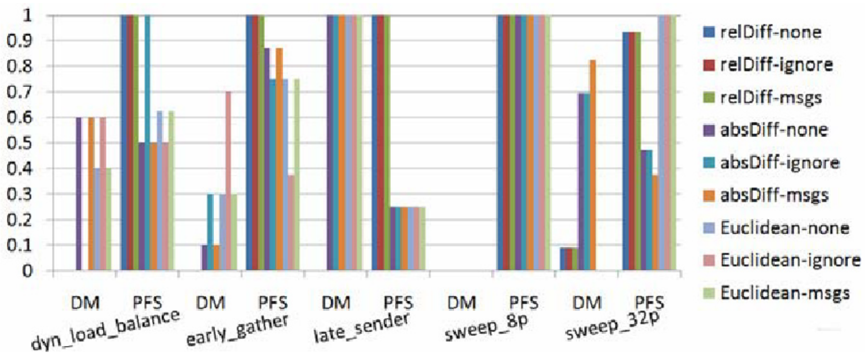


Fig. 3. Percentage File Size and Degree of Matching. Here we show percentage file size (PFS) degree of matching (DM). The thresholds used, in order of the bars, were: 0.8, 0.8, 0.8, 10^4, 10^3, 10^4, 0.6, 0.4, 0.6, 0.8, 0.8, 0.8, 10^4, 10^4, 10^4, 0.1, 0.1, 0.1, 0.8, 0.8, 0.8, 10^6, 10^6, 10^6, 1.0, 0.6, 1.0, 0.8, 0.8, 0.8, 10^6, 10^6, 10^5, 2.0, 2.0, 1.2, 0.98, 0.98, 0.98, 10^6, 10^6, 10^5, 1.4, 1.4, 1.4.

neutron transport problem [1]. We collected traces for two runs of this application. The 8-process run, sweep3d_8p, used the input file input.50 from the application distribution; the 32-process run, sweep3d_32p, used the input.150 file.

Distance Metrics. The distance metrics we use to compare trace segments are the relative difference (*relDiff*), absolute difference (*absDiff*), and Euclidean distance (*Euclidean*) at different thresholds.

relDiff. We compare the relative differences between measurements against a threshold; if greater, the segments are not equal. The formula we use is: $relDiff = (|x_1\text{-}x_2|)/max(x_1,x_2)$. To see how *relDiff* matches segments, consider our example in Fig. 2 using a threshold of 0.4. To compare *s0.0* with *s0.1*, we begin with the start times of the segments: x_1=25 and x_2=24, and compute a relative difference of 0.04. Since the relative difference is less than 0.4, we check the start times of the `do_work` event: x_1=1 and x_2=1, relative difference 0. We continue checking the timings pairwise until we reach the end of the segment. If the segments match, we move on to compare the next segments, *s1.0* and *s1.1*. The end times of `do_work` are x_1=40 and x_2=23, with relative difference 0.42. This is above our threshold, so the segments do not match, and therefore the match fails for the traces of ranks 0 and 1.

absDiff. As with the *relDiff*, each measurement is compared with its counterpart. A fixed size difference, determined by a threshold, is allowed for each measurement pair. Using our example segments in Fig. 2, and a threshold of 15, we see that *s1.0* will not match *s1.1*, because the end times of `do_work` are 17 time units apart.

Euclidean. We compute the Euclidean distance between measurements in segments and compare it to a threshold multiplied by the maximum value in the measurements. Using our example in Fig. 2, to compare *s1.0* and *s1.1*, we create a vector of the measurements for *s1.0*, (75, 1, 40, 41, 50, 51), and for *s1.1*, (76, 1, 23, 24, 54, 55) and compute a distance of 24.7 between them. The largest measurement in the pair of vectors is 55. Given a threshold 0.2, then the highest the computed distance can be for a match is 11. Thus, the traces for rank 1 and rank 0 do not match.

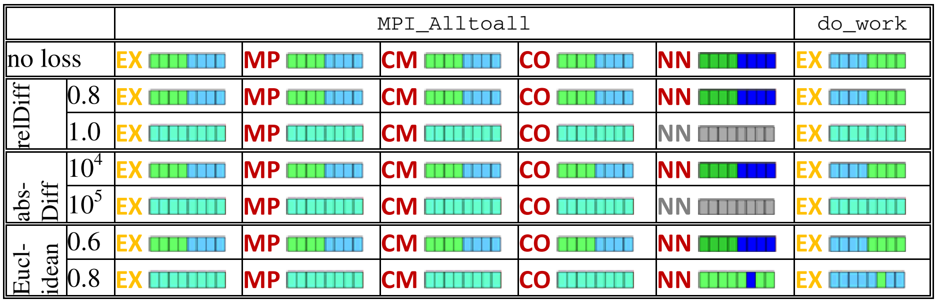


Fig. 4. Performance Trends for dyn_load_balance with No Relaxation. Each bar represents the severity of the performance problem from low (blue) to high (red), or white for 0. Each box within the bar represents one MPI rank. We abbreviate the diagnoses: EX: Execution, MP: MPI, CM: Communication, CO: Collective, NN: Wait at NxN.

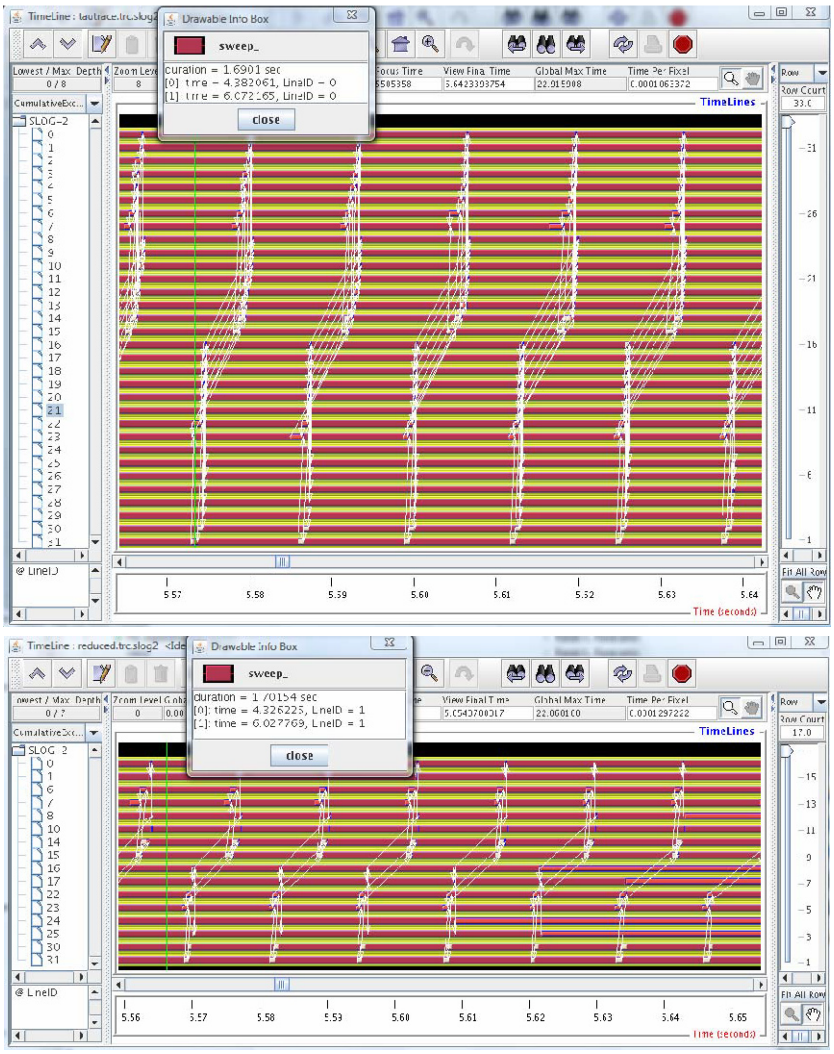
5 Experimental Results

We evaluate the matching criteria based on amount of trace matching, error in the trace, and whether the reduced trace visualization leads to the same performance conclusion as a visualization of a full trace.

Trace Matching. In Fig. 3 we present the percentage of original trace file size and degree of matching achieved when reducing the traces using our matching methods and relaxation strategies. Considering only the event stream and not the behavior of the programs, the highest possible matches for each are: 7 for dyn_load_balance and early_gather; 6 for late_sender; 0 and 16 for sweep3d_8p and sweep3d_32p, respectively, with no relaxations, and 2 and 23 when relaxing message passing checks, respectively. We found that *relDiff* was not able to find any acceptable matches for all but sweep3d_32p. *AbsDiff* found acceptable matches for all but sweep3d_8p and for dyn_load_balance when ignoring segments. For sweep3d_32p, when relaxing message passing parameter checks, *absDiff* was able to find more matches and still retain performance trends. The *Euclidean* method was able to find acceptable matches for all but the two sweep3d runs. When *Euclidean* was able to find matches for sweep3d, they were always unacceptable.

Observing the results for the different relaxation strategies, we see that there seems to be benefit for ignoring segments for dyn_load_balance with *Euclidean* and early_gather with *absDiff* and *Euclidean*. The sweep3d_32p program benefitted from relaxing message passing checks with *absDiff*, but suffered from the relaxation with *relDiff*, because all matches found were unacceptable.

Trace Error. We computed the approximation distance for each method, relaxation, and program. Interestingly, we found that the highest errors were introduced into the late_sender program, for which all methods but *relDiff* were able to find matches that still retained performance trends. High errors were also introduced into sweep3d_32p for *relDiff* and *absDiff*, while still retaining correct performance trends. For early_gather, high error was introduced by *absDiff* and *Euclidean* when ignoring initialize and finalize segments.


TimeLine : tautrace.trc.slog2

Drawable Info Box

sweep_

duration = 1.6901 sec
[0]: time = 4.382061, LineID = 0
[1]: time = 6.072165, LineID = 0

Lowest / Max Depth: 0 / 8
Zoom Level: 8
Focus Time: 5505358
View Final Time: 5.6423393754
Global Max Time: 22.015908
Time Per Pixel: 0.0001063372
Row Count: 33.0

TimeLine : reduced.trc.slog2

Drawable Info Box

sweep_

duration = 1.70154 sec
[0]: time = 4.326225, LineID = 1
[1]: time = 6.027769, LineID = 1

Lowest / Max Depth: 0 / 7
Zoom Level: 0
View Final Time: 5.6540700017
Global Max Time: 22.060100
Time Per Pixel: 0.0001297222
Row Count: 17.0

Fig. 5. Trace Visualization of sweep3d_32p. The top shows the complete set of traces. The bottom shows the reduction from *absDiff* at 10^6 with no relaxation.

Retention of Trends. We examined the reduced traces for the methods and relaxation strategies at differing thresholds. We show an example in Fig. 4. The first line in Fig. 4 is the results for the original, complete trace. We show the acceptable (top bars) and unacceptable (bottom bars) matches for each method with no relaxation for dyn_load_balance in Fig. 4. *RelDiff* only found matches at a threshold of 1.0, but lost performance trends. *AbsDiff* and *Euclidean* both found matches that retained trends, at thresholds 10^4 and 0.6, and degrees of matching of 0.6 and 0.4, respectively.

In Fig. 5, we show example trace visualizations of sweep3d_32. The top screenshot shows the complete set of process traces. The bottom shows the set of representative traces from *absDiff* at threshold 10^6 with no relaxation. From this, we can see the groups of processes that had the same performance and message passing behaviors. In order to understand the exact message passing patterns, we may need to provide additional information in the visualization, such as the pattern of communication being shown, e.g. ranks send to rank+1 in a nearest neighbor communication pattern.

Discussion. Each of the methods behaved differently when matching traces. We found that *relDiff* matched nothing except at the highest threshold. This occurred because the relative difference between small time stamps caused matching failures, e.g. for time stamps s1=1 and s2=0, the relative difference is 1.0 even though there is only 1 time unit between them. Because any amount of error was allowed at threshold 1.0, the matching was very aggressive and the trends were not retained in the reduced trace. For all programs except sweep3d_32p, *relDiff* failed to produce an acceptable match. *AbsDiff* did well on all programs but sweep3d_8p and dyn_load_balance. When ignoring segments with dyn_load_balance, *absDiff* allowed five matches instead of just the optimal four, causing a loss of performance trends. Four matches is optimal because of the eight processes in the run, two groups of 4 behaved similarly. *Euclidean* did well on the benchmarks, but, for sweep3d, found only matches in which trends were lost.

6 Conclusions

In this study, we examined methods for inter-process trace reduction for the purpose of visualizing application performance. Although none of the methods performed perfectly, we conclude that *absDiff* performed the best, because it was able to find the most acceptable matches for the largest number of program traces. We found that it is indeed possible to reduce the number of trace lines in a visualization, while deducing the same performance diagnosis as when examining the full set of traces. Ongoing directions for this work include examination of other distance methods for deciding trace matching; investigating other relaxation strategies that could increase matching while not losing important diagnostic information; a study of larger-scale applications; and exploring information that could help clarify message passing patterns in reduced traces.

References

[1] The ASCI sweep3D readme file (January 2009), `http://www.c3.lanl.gov/pal/software/sweep3d/sweep3d_readme.html`

[2] Aguilera, M.G., Teller, P.J., Taufer, M., Wolf, F.: A systematic multi-step methodology for performance analysis of communication traces of distributed applications based on hierarchical clustering. In: IPDPS (2006)

[3] Fahringer, T., Gerndt, M., Mohr, B., Wolf, F., Riley, G., Traff, J.: Knowledge specification for automatic performance analysis. Technical Report Revised Edition, ESPRIT IV Working Group on Automatic Performance Analysis: Resources and Tools APART (January 2001), `http://www.fz-juelich.de/apart-1/reports/wp2-asl.ps.gz`

[4] Freitag, F., Caubet, J., Labarta, J.: A trace-scaling agent for parallel application tracing. In: Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02), Washington, DC, USA, p. 494. IEEE Computer Society, Los Alamitos (2002)

[5] Geimer, M., Wolf, F., Wylie, B.J.N., Mohr, B.: Scalable parallel trace-based performance analysis. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 303–312. Springer, Heidelberg (2006)

[6] Gerndt, M., Mohr, B., Träff, J.L.: A test suite for parallel performance analysis tools. Concurrency and Computation: Practice and Experience 19(11), 1465–1480 (2007)

[7] Hackstadt, S., Malony, A., Mohr, B.: Scalable performance visualization for data-parallel programs, May 1994, pp. 342–349 (1994)

[8] Heath, M., Malony, A., Rover, D.: The visual display of parallel performance data. Computer 28(11), 21–28 (1995)

[9] Heath, M.T., Etheridge, J.A.: Visualizing the performance of parallel programs, vol. 8, pp. 29–39. IEEE Computer Society Press, Los Alamitos (1991)

[10] Hollingsworth, J., Miller, B., Cargille, J.: Dynamic program instrumentation for scalable performance tools. In: Proceedings of Scalable High Performance Computing Conference, Knoxville, TN, USA, May 23-25, pp. 841–850 (1994)

[11] Knüpfer, A., Voigt, B., Nagel, W.E., Mix, H.: Visualization of repetitive patterns in event traces. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, pp. 430–439. Springer, Heidelberg (2007)

[12] Lee, C.W., Mendes, C., Kalé, L.V.: Towards scalable performance analysis and visualization through data reduction. In: 13th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2008) held in conjunction with IPDPS (2008)

[13] Miller, B.P.: What to draw? when to draw?: an essay on parallel program visualization. J. Parallel Distrib. Comput. 18(2), 265–269 (1993)

[14] Mohror, K., Karavanic, K.L.: Towards scalable event tracing for high-end systems. In: Perrott, R., Chapman, B.M., Subhlok, J., de Mello, R.F., Yang, L.T. (eds.) HPCC 2007. LNCS, vol. 4782, pp. 695–706. Springer, Heidelberg (2007)

[15] Mohror, K., Karavanic, K.L.: Evaluating similarity-based trace reduction techniques for scalable performance analysis. In: SC '09: Proceedings of the 2009 ACM/IEEE conference on Supercomputing (2009)

[16] Naím, O., Hey, A.J.G.: Visualization of do-loop performance. In: HPCN Europe, pp. 878–887 (1997)

[17] Nickolayev, O., Roth, P., Reed, D.: Real-time statistical clustering for event trace reduction. International Journal of High Performance Computing Applications 11(2), 69–80 (1997)

[18] Noeth, M., Mueller, F., Schulz, M., de Supinski, B.R.: Scalable compression and replay of communication traces in massively parallel environments. In: 21th International Parallel and Distributed Processing Symposium IPDPS'07 (March 2007)

[19] Ratn, P., Mueller, F., de Supinski, B.R., Schulz, M.: Preserving time in large-scale communication traces. In: ICS '08: Proceedings of the 22nd annual international conference on Supercomputing, pp. 46–55. ACM, New York (2008)

[20] Song, F., Wolf, F., Bhatia, N., Dongarra, J., Moore, S.: An algebra for cross-experiment performance analysis. In: Proc. of the International Conference on Parallel Processing (ICPP), Montreal, Canada, August 2004, pp. 63–72. IEEE Society, Los Alamitos (2004)

[21] Zaki, O., Lusk, E., Gropp, W., Swider, D.: Toward scalable performance visualization with Jumpshot. The International Journal of High Performance Computing Applications 13(3), 277–288 (Fall 1999)

Workshop on Real-Time Interactive Applications on the Grid (ROIA 2009)

Preface

Through recent advancements in network technologies, graphics cards and displays, a new type of Real-time Online Interactive Applications (ROIA) has become increasingly popular. Everyday life is currently being affected and transformed not only by the use of Web technologies, but also by collaborative multimedia applications, networked computer games, cooperative scientific visualisations, networked virtual environments and real-time graphics displays. Computer-Supported Cooperative Work (CSCW) and Massively Multiplayer Online Gaming (MMOG) are two huge growing sectors worldwide with challenging demands with respect to real-time distributed and interactive technologies.

ROIA 2009 is the second edition of this workshop organised in conjunction with the Euro-Par conference at the Technical University of Delft, Netherlands. The focus of the workshop is on all areas of real-time distributed technologies, from research of basic real-time methods, to applications in real-world environments. The ROIA workshop has offered possibilities to discuss the benefits of real-time applications for human users, to show the latest results, products or research prototypes, and to establish connection between developers and users of associated technologies.

The topics of interest discussed at the workshop included real-time interactive parallel and distributed tools and environments, real-time interactive distributed (massively multiplayer) online gaming, real-time interactive e-learning applications, integration of Cloud computing virtualisation technologies with real-time interactive applications, techniques for real-time Quality of Service (QoS) monitoring and enforcement, utility business models and Service Level Agreements (SLA) for ROIAs, and experiences in deployment and use of real-world distributed ROIAs.

This year's workshop organised by the FP6 IST-034601 edutain@grid STREP project consortium accepted six technical papers and two tutorials scheduled over two half days. To ensure high quality, each paper underwent two rounds of reviews carried out by at least three international experts.

The first day has been dedicated to presentations regarding the edutain@grid project that targeting real-time scalability, resource management and business support for ROIA in Grid environments.

The first paper by Stuart Middleton et al. titled "Bipartite Electronic SLA as a Business Framework to Support Cross-Organisation Load Management of Real-Time Online Applications" presents a novel Grid-based business framework that makes use of bipartite SLAs and dynamic invoice models to model complex business relationships in a massively scalable and flexible way. For evaluation it looks at existing and extended value chains, the QoS metrics measured and the dynamic invoice models that support this work. The causal links from customer quality of experience (QoE) and service provider quality of business (QoBiz) through to measured quality of service are examined. Finally it discusses a shared reward business ecosystem and suggests how extended SLAs and invoice models can support this.

The second paper by Vlad Nae et al. titled "Monitoring and Fault Tolerance for Real-Time Online Interactive Applications" presents a monitoring system which collects data from all resources in a distributed environment and from the ROIA managed by the edutain@grid platform. It also describes a fault tolerance service which addresses not only the faults commonly encountered in distributed systems, but also faults manifesting at service level, within the platform's management services. Finally, a use-case consisting of the platform running a MMOG as a concrete ROIA demonstrates the roles of the monitoring and fault tolerance services.

The third paper by Frank Glinka et al. titled "A Service-Oriented Interface for Highly Interactive Distributed Applications" describes a service-oriented interface that comprises a Real-Time Framework (RTF) supporting a high-level application development process which frees the software developer from the low-level details of distributed computation and communication, and the Hoster Management Interface (HMI) supporting transparent resource management for a running application, in particular the creation, controlling and monitoring of ROIA instances. The paper presents an efficient implementation of the interface and describes its use for two particular distributed application scenarios.

The first day of the workshop concluded with a tutorial by Alexander Ploß et al. on "Scaling Real-Time Games on Grid Resources". The tutorial addressed had two goals targeted towards two demographic audiences, namely real-time application developers and edutain@grid platform hosters, both being platform-users, but as different business actors. The first goal of the tutorial was an introduction to the utilisation of the RTF library, enabling real-time applications to be managed within edutain@grid. The second target familiarised the participants with the edutain@grid management services and their functionality accessed through a special purpose management portal. Both parts contained live demonstrations of the involved concepts.

The workshop continued on the next day with a tutorial by Stuart Middelton et al. on "Dynamic SLA, QoS Measurement and Invoicing Support for ROIA in edutain@grid" that showed how the edutain@grid business layer framework supports the three stages of SLA management, contract definition, negotiation, and enforcement, in a way suitable for ROIA requirements.

The first technical paper of the second day by Eryk Ciepiela et al. titled "CompTalks – From a Meta-Model Towards a Framework for Application-Level Interaction Protocols" introduces a new concept of conversation protocol to enable custom fine-grained and elaborate message exchange between distributed yet tightly-coupled parties. The framework is successfully applied to develop a protocol for GSEngine which serves as the runtime system of the ViroLab virtual laboratory, enabling development and execution of complex collaborative applications.

The second paper by Alexandru Iosup titled "CAMEO: Continuous Analytics for Massively Multiplayer Online Games on Cloud Resources" introduces a new architecture that provides various mechanisms for MMOG data collection and continuous analytics of a pre-determined accuracy in real settings. The paper assess the capabilities of the proposed approach by taking and analysing complete or partial snapshots from Runescape, one of the most popular MMOGs with a community of over 3,000,000 active players. Notably, it shows evidence that CAMEO already supports simple continuous MMOG analytics, and give a first estimation of the costs of the analytic process.

The last paper presented at the workshop by Tomasz Jaskiewicz titled "Complex Multiplayer Urban Design System – Concept and Case Studies" explores the idea of creating a software and hardware system supporting collaborative urban planning and design. The paper demonstrates several working case studies of various parts of such system and illustrated a selected strategy for a computer supported cooperative work for the field of architectural and urban design and planning.

As Program Chair, I wish to acknowledge all those that contributed to the success of ROIA 2009, in particular to the authors of the submitted papers, and to the Program Committee members for their valuable time and expertise to the selection process.

December 1, 2009

Radu Prodan
Program Chair
ROIA 2009

Bipartite Electronic SLA as a Business Framework to Support Cross-Organization Load Management of Real-Time Online Applications

Stuart E. Middleton, Mike Surridge, Bassem I. Nasser, and Xiaoyu Yang

IT Innovation Centre, University of Southampton,
2 Venture Road, Southampton, SO16 7NP, UK
{sem,ms,bmn,kxy}@it-innovation.soton.ac.uk

Abstract. Online applications such as games and e-learning applications fall within the broader category of real-time online interactive applications (ROIA), a new class of 'killer' application for the Grid that is being investigated in the edutain@grid project. The two case studies in edutain@grid are an online game and an e-learning training application. We present a novel Grid-based business framework that makes use of bipartite service level agreements (SLAs) and dynamic invoice models to model complex business relationships in a massively scalable and flexible way. We support cross-organization load management at the business level, through zone migration. For evaluation we look at existing and extended value chains, the quality of service (QoS) metrics measured and the dynamic invoice models that support this work. We examine the causal links from customer quality of experience (QoE) and service provider quality of business (QoBiz) through to measured quality of service. Finally we discuss a shared reward business ecosystem and suggest how extended service level agreements and invoice models can support this.

Keywords: SLA, business model, value chain, cross-organization, load management, ROIA, Grid.

1 Introduction

As Grid technology matures [8] it raises the possibility of improving the way that online applications such as games and e-learning applications are provisioned and managed. The edutain@grid project [7] is investigating just this. This type of application needs resource provisioning that is secure, robust, scalable and flexible enough to support the value chains found in real-time online domains. As case studies within the edutain@grid project we have two distinct ROIAs, a real-time massively multiplayer online (MMO) game developed by Darkworks and an e-learning search and rescue training simulator developed by BMT Cordah. Through these case studies we aim to evaluate how Grid technology can support and provision ROIAs and their associated business relationships.

The online game market sector is growing, soon to be worth billions [5], and the e-learning market is currently worth millions [10]. Analysis of the business relationships is key to developing a commercially viable supporting middleware. In edutain@grid we have implemented a business layer that flexibly supports complex value chains in a way where multi-organizational resource provision can scale massively and gracefully with ROIAs as they become more successful and attract more customers. Extending an existing business Grid middleware, GRIA [17], we make use of bipartite service level agreements (SLAs) and dynamic invoice models to encode business relationships. Although not the focus of this paper, our middleware supports single sign-on security, with X.509 credentials and Security Assertion Markup Language (SAML) access control tokens put in place prior to user's game play to avoid real-time performance costs.

This paper presents our novel business framework, using scalable dynamic bipartite service level agreements and invoice models based on quality of service. Our business level support for cross-hoster load management, though zone migration, is not currently seen with ROIA provisioning today. The concept of zone depends on the application and can be 3D areas in a game world, training scenarios etc. In addition to our proof of concept implementation we present a new shared reward business ecosystem that could help shape ROIA provisioning models as they grow in scale over the coming years.

2 Related Work

Most Grid middleware systems such as the Globus toolkit [9], gLite [6], and UNICORE [3] have somewhat rigid infrastructures and are not very cost-effective at supporting changes to the basic business infrastructure associated with a dramatic scaling-up of service provision requirement. In edutain@grid our support for bipartite business relationships makes the provisioning network flexible and easy to grow over time. Supporting cross-hoster service provision and load management by design allows us to manage the changes in ROIA scale cost-effectively.

The use of service level agreements has been used as part of the paper management of supply chains and telecommunication services for decades. As service provision becomes more dynamic, with increasingly agile service composition, electronic service level agreement lifecycle management gains importance. A number of standardization attempts have been seen [13] but failed to gain traction within the community (e.g. WLSA, SLAng). Currently WS-Agreement [1] is the most widely adopted standard to represent service level agreements, but focuses on protocol and lacks detailed standards for representing quality of service metrics, constraints and penalties. Edutain@grid builds on this work defining bipartite service level agreements between coordinators and hosters to model our business relationship networks in a flexible way, and to set quality of service expectations from ROIA provisioning that can be measured and monitored.

The associated area of cloud computing has come about from an evolution of grids and service oriented architectures [18] and gained popularity when IBM and Google [12, 14] announced their collaboration. Clouds focus on virtualization coupled with time / CPU multiplexing, load balancing and multi-user service hosting to provide scalability. The cloud middleware hides the complexity involved in finding and preparing remote 'bare metal' computing resources. Cloud computing is a scalable solution but current implementations ignore geographic location (important for network performance), are single-hoster and lack support for dynamic service level agreements [4]. In edutain@grid we support load balancing between multiple hosters and use bipartite service level agreements to manage complex business relationships.

In the gaming space there are a number of existing commercial implementations of middleware for large scale 3D worlds supporting massive multiplayer online games. The Grid community has had some impact into this area with commercial offerings from Butterfly Grid [11] and BigWorld [2]. Butterfly Grid is based on the Globus toolkit and provides a peer to peer network of servers at a single hoster along with IP level security and single sign-on for in-game user accounts. BigWorld server provides single-hoster cluster management along with zone migration and bandwidth control via level of detail prioritization. Edutain@grid moves beyond these capabilities by supporting multiple hosters, and cross-hoster load management through zone migration, allowing massive scale-up to gracefully occur around successful ROIAs.

For the e-learning sector frameworks [16] have been developed using client-server, peer to peer and web service architectures but all suffer from associated poor scalability and fault tolerance / reliability. More recently Grid technology has been introduced in an attempt to bring in scalable distributed resources and allow e-learning applications with higher resource demands to be developed cost-effectively. This Grid focus is on automated service composition and adaption. In edutain@grid we support e-learning applications with real-time performance criteria, a new aspect that has not been applied to e-learning Grids yet.

3 Real-Time Online Interactive Application Case Studies

The edutain@grid project includes two exemplar case study applications; an online multiplayer game and an e-learning multi-student training application. These applications have allowed the edutain@grid project to build a proof of concept architecture and test different aspects of our approach to ROIA provisioning. Figure 1 provides screenshots from these applications in action.

A massively scalable online 3D first person cooperative shoot-em-up game has been developed by Darkworks called ‘Hunter’. This is a fast paced game with a massively scalable 3D hexagonal segmented play area that grows as new players connect. Key quality of service metrics are client packet latency (<500ms) and server frame rate (>15 frames/sec). This pilot application is typical of massively multi-player online (MMO) first person perspective (FPS) games where games support 1000’s of players from multiple geographic regions.

An e-learning shell application has been developed by BMT Cordah to run training applications such as their Search and Rescue (SAR) application within a multi-user voice over IP (VOIP) support environment. Supervisors and students remotely connect, and control is shared via a hot-seat protocol. The supervisor can monitor each student’s progress as they participate in coast-guard role-play training simulations and communicate using video and audio. The application supports the Sharable Content Object Reference Model (SCORM) standard in common with most commercial e-learning applications. Sessions can involve up to 100 students and a few supervisors from multiple geographic regions. The key quality of service metric is data throughput to ensure acceptable VOIP performance during training sessions.

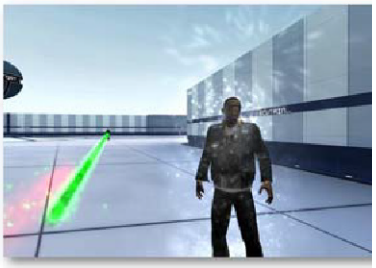
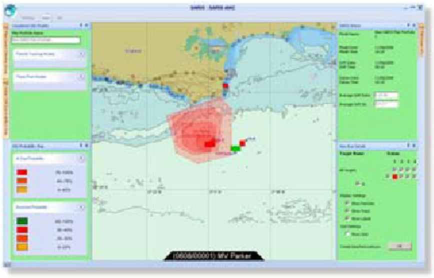



Fig. 1. Hunter online game and Search and Rescue (SAR) e-learning application screenshots

4 Flexible Business Models Suitable for ROIA

The value chain for existing commercial ROIA provision is relatively simple, with a single service provider, or ‘hoster’, provisioning model underpinned by written fixed term service level agreements, between a customer and the hoster. This service level agreement defines the hardware that will be provided for the duration of the contract and cost to the customer. Penalties are often written in to compensate for failure of hardware availability or uptime. The scalability of this type of provisioning model is limited to the number of servers a hoster can provide. Vendor lock-in is a restriction for customers, and for ROIAs with large user-bases in many geographic regions multiple vendor agreements are often needed to ensure servers are geographically close to clients to increase communication performance.

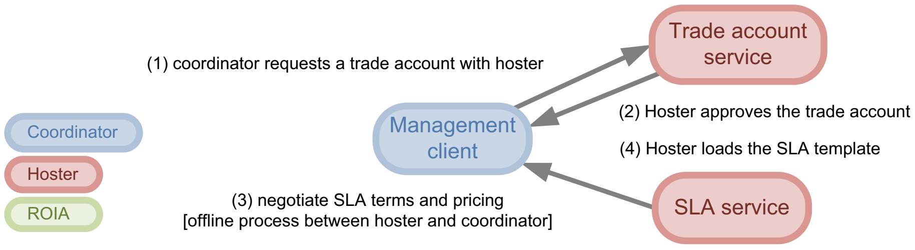
In edutain@grid we have experimented with bringing Grid concepts to support more scalable multi-hoster value chains. We recognize that ROIA provision needs to start small, with an entry level low-cost single-hoster provision, and scale up gracefully through several orders of magnitude of users as a ROIA grows in success and popularity. We have introduced a third actor into the current commercial provisioning relationship, a broker or ‘coordinator’, that allows flexible value chains made up of many on-demand bipartite business relationships. The customer, or game player, is assigned a server provisioned by a hoster via the coordinator. We use on-demand electronic bipartite service level agreements to encode pricing and expected quality of service between the coordinator and hoster. User account management is provided by the coordinator for the customer. Hosters run a trade account service to record invoices for provisioned service and coordinators make use of existing customer payment models (e.g. PayPal).

Electronic on-demand service level agreements allow pricing based on measured quality of service and resource usage, not just hardware costs. This flexibility to pay for what is actually used allows coordinators to start small, sharing hoster resources with other coordinators. As users for a ROIA increase a greater share of each hoster’s resource can be taken and new hosters brought in to provision the increased load. Edutain@grid supports cross-hoster zone migration allowing seamless load balancing between hosters with differing resource available from different geographic regions.

Supporting the electronic service level agreements in edutain@grid is a flexible invoice model that provides variable pricing, with cost components proportional to the quality of service measured, and banded pricing, where the overall cost is linked to bands based on quality of service threshold levels achieved by the provider. Classic invoicing components are also provided for cost per duration and penalty fees for breaches of quality of service thresholds. These invoice tools provide us with a flexible business layer that supports a variety of mechanisms to provide business incentives for key actors in the ROIA value chain.

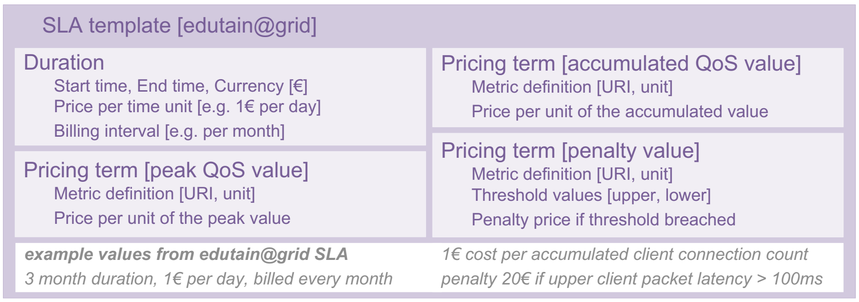
5 Case Study: edutain@grid Business Layer Architecture

The edutain@grid business layer implementation supports the three phases of the service level agreement lifecycle, contract definition, negotiation and enforcement. For contract definition we have implemented a workflow, shown in figure 2. Multiple service level agreements can be setup for multiple coordinators and ROIAs providing a flexible and scalable bipartite value network.



Trade account service
(1) coordinator requests a trade account with hoster
Coordinator
Hoster
ROIA
Management client
(2) Hoster approves the trade account
(4) Hoster loads the SLA template
(3) negotiate SLA terms and pricing [offline process between hoster and coordinator]
SLA service

Fig. 2. edutain@grid SLA contract definition workflow



SLA template [edutain@grid]
Duration
Start time, End time, Currency [€]
Price per time unit [e.g. 1€ per day]
Billing interval [e.g. per month]
Pricing term [accumulated QoS value]
Metric definition [URI, unit]
Price per unit of the accumulated value
Pricing term [peak QoS value]
Metric definition [URI, unit]
Price per unit of the peak value
Pricing term [penalty value]
Metric definition [URI, unit]
Threshold values [upper, lower]
Penalty price if threshold breached
example values from edutain@grid SLA
3 month duration, 1€ per day, billed every month
1€ cost per accumulated client connection count
penalty 20€ if upper client packet latency > 100ms

Fig. 3. edutain@grid XML SLA template outline

The edutain@grid service level agreement XML template structure, figure 3, contains sections for static hardware provision, cost for duration of provision, variable cost components based on quality of service measurement and penalties based on breaches of agreed thresholds. We use metrics for server packet latency (ms), packet loss (%), data throughput (bytes/s), server tick time (ms) and client connection count.

We have implemented a discrete offer protocol for contract negotiation (figure 4) in addition to session management; hosters have local provisioning sessions and coordinators have global sessions to manage collections of local session. More complex multi-stage negotiation strategies are possible but not cost effective for the value of individual provisioning contracts (typically €100's for a few months).

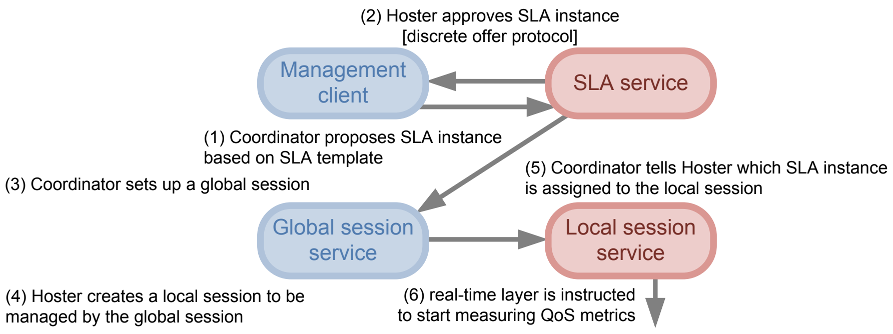

(2) Hoster approves SLA instance
[discrete offer protocol]
Management client
SLA service
(1) Coordinator proposes SLA instance based on SLA template
(3) Coordinator sets up a global session
(5) Coordinator tells Hoster which SLA instance is assigned to the local session
Global session service
Local session service
(4) Hoster creates a local session to be managed by the global session
(6) real-time layer is instructed to start measuring QoS metrics

Fig. 4. edutain@grid SLA contract negotiation workflow

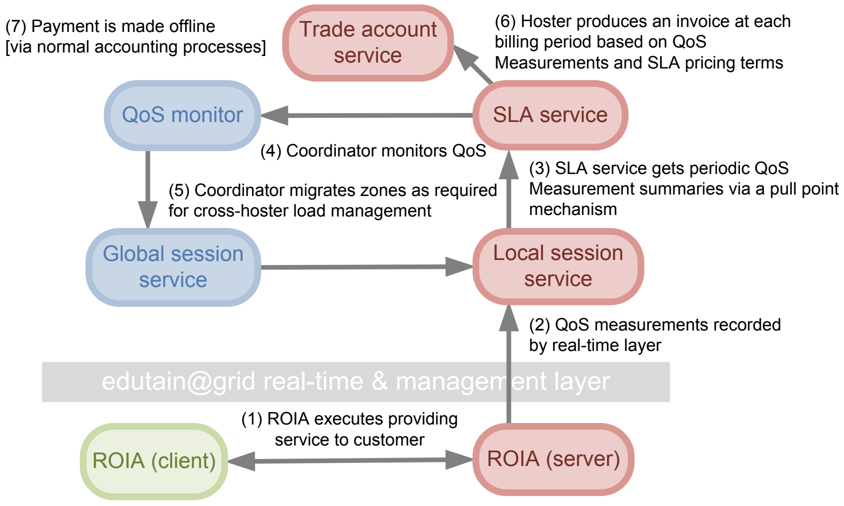

(7) Payment is made offline
[via normal accounting processes]
Trade account service
(6) Hoster produces an invoice at each billing period based on QoS Measurements and SLA pricing terms
QoS monitor
SLA service
(4) Coordinator monitors QoS
(5) Coordinator migrates zones as required for cross-hoster load management
(3) SLA service gets periodic QoS Measurement summaries via a pull point mechanism
Global session service
Local session service
(2) QoS measurements recorded by real-time layer
edutain@grid real-time & management layer
(1) ROIA executes providing service to customer
ROIA (client)
ROIA (server)

Fig. 5. edutain@grid contract enforcement workflow

The final step is contract enforcement, shown in figure 5, where users join ROIA sessions and quality of service metrics are recorded for the duration of each session runtime. The coordinator will monitor quality of service levels and can choose to (manually and/or automatically) migrate zones from one hoster's session to another, allowing cross-hoster load balancing. Edutain@grid thus implements business level control over real-time hoster to hoster zone management, something not seen in ROIAs today. The invoice is based on the terms in the service level agreement and actual payment by the coordinator to the hoster is made via normal accounting procedures.

6 Evaluation beyond Quality of Service for ROIA Provision

If we look at the whole business eco-system [15] we see that quality of service (QoS) are objective facts that are measureable, but what really matters to actors in the ecosystem depends on their perspective. The customer is primarily interested in the quality of experience (QoE) that good quality of service allows, ensuring the ROIA delivers as expected. The coordinator and hoster are interested in the quality of business (QoBiz), in particular the value gained for doing their role in service provision. If the QoE and QoBiz are causally linked to measureable QoS then the business ecosystem as a whole should be able to prosper.

From the customers perspective QoE for a game is linked to the ability to connect to a server, play with friends, ease of connection and use of the ROIA and the lack of any game perceivable game lag. For an e-learning application QoE means the ability to connect to server, talk to the supervisor and the quality of coaching received. There is a relatively clear cause and effect from QoE to the key QoS metrics. The data throughput will affect the ability of students to understand the supervisor via VOIP. The server frame rate and client packet latencies will affect game lag. Simple single sign-on security ensures easy login.

The impact QoE has on the value chain is on customer repeat business and the likelihood of attracting new business through word of mouth. These effects will impact future customer numbers, and thus the value of the overall business proposition for a ROIA provision network.

From the coordinator and hosters perspective the QoBiz comes down to the revenue obtained from the business proposition. Each decision they must make is done so in the context of how it will affect their QoBiz. For hosters key decisions are:

- will they accept new load
- will they signal to the coordinator they are (or might be) having trouble provisioning existing load
- are they able to shift internal resource to ensure QoS for existing load
- how much to charge a coordinator for provision of service.

For coordinators key decisions are :

- which hoster (who, where) should receive new load
- when, where and who to migrate ROIA load cross-hoster
- if, when and where a new hoster should be brought into the scalable value network for a specific ROIA
- how much to charge a customer for using a ROIA
- how much to pay hosters for service provision.

In order to ensure good QoBiz pricing incentives must be associated with each key business decisions and ultimately causally linked back to the final customers QoE. In this way a value chain and associated business model is setup so that all stakeholders are incentivised to increase overall QoBiz. Figure 6 shows the business ecosystem from a QoBiz perspective, showing actors and how revenue flows between them.

We have investigated within the edutain@grid business layer implementation pricing instruments for hardware prices for a duration, variable prices per quality of service measurement, penalty costs for quality of service breaches and price banding.

A hardware cost per duration incentivises the hoster to accept load at every opportunity. A penalty cost reduces the incentive to under-provision and provides a basic incentive framework in which ROIAs can be provisioned. However there is no incentive for the hoster to work to provide better quality of experience, or help grow the quality of business; the only incentive is to provide momentary quality of service on a case by case basis.

Introducing variable pricing based on measured quality of service allows us to define customer focused metrics such as the number of client connections, server frame rate, client connection latency and data throughput. Banded pricing provides an increasing scale of penalty for bad quality of service and helps to discourage systematic under-provisioning that would otherwise be in the hosters interest since it would ensure resources are fully loaded at all times. These incentives link hoster provisioning to factors that affect customer quality of experience. It is up to the coordinator to select carefully the key quality of service metrics that really do have a causal link back to QoE; this might be difficult if the causal link is not clear. Improving QoE is likely to indirectly improve the QoBiz, via long term return business, so the coordinator is well motivated to ensure this.

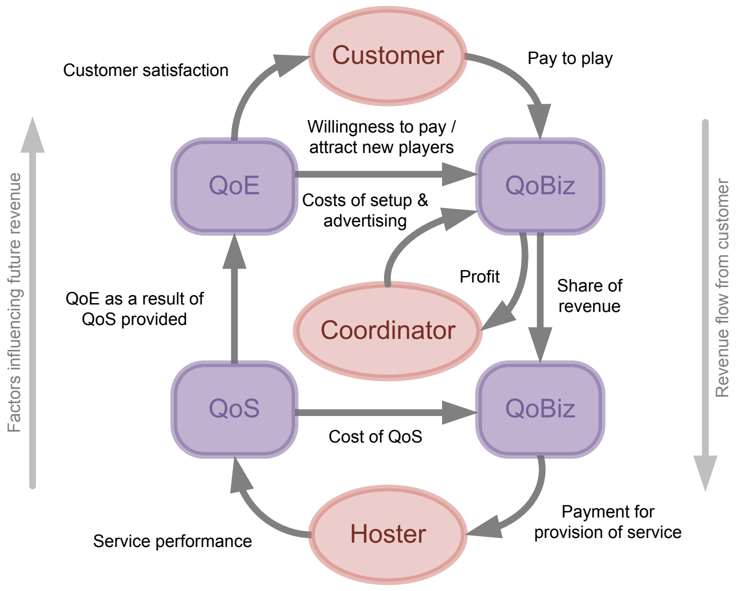

Customer
Customer satisfaction
Pay to play
Willingness to pay / attract new players
QoE
QoBiz
Costs of setup & advertising
Profit
Share of revenue
QoE as a result of QoS provided
Coordinator
Factors influencing future revenue
Revenue flow from customer
QoS
QoBiz
Cost of QoS
Service performance
Hoster
Payment for provision of service

Fig. 6. Business ecosystem for actors in the ROIA value chain

We have found from experience in edutain@grid that these techniques are the limit with which the service and ROIA providers are really commercially comfortable, being not too far from the existing single-hoster fixed contract provisioning models that work commercially today. These pricing instruments do not, however, give hosters any direct incentive to work together to that ensure cross-hoster QoE is maintained.

We envisage a further enhancement to this incentive framework where the revenue from customers is directly shared, via coordinators, with the hosters. This shared value network has the advantage that hosters have a direct incentive to see the QoBiz grow. A service level agreement could define variable quality of service rewards in terms of a percentage of the revenue obtained from each customer, with a banded reward adjustment based on a longer term business metrics such as player number growth or increased coordinator income. Such shared rewards should encourage a limited degree of cooperation between hosters, encouraging proactive load sharing for under-provisioned hosters. Figure 7 shows what a shared incentive service level agreement might look like.

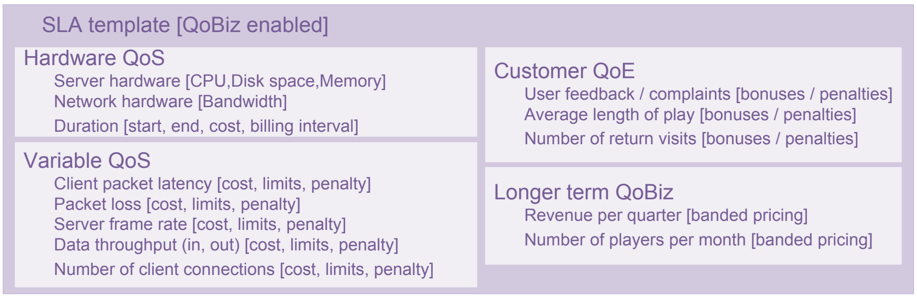

SLA template [QoBiz enabled]
Hardware QoS
Server hardware [CPU,Disk space,Memory]
Network hardware [Bandwidth]
Duration [start, end, cost, billing interval]
Variable QoS
Client packet latency [cost, limits, penalty]
Packet loss [cost, limits, penalty]
Server frame rate [cost, limits, penalty]
Data throughput (in, out) [cost, limits, penalty]
Number of client connections [cost, limits, penalty]
Customer QoE
User feedback / complaints [bonuses / penalties]
Average length of play [bonuses / penalties]
Number of return visits [bonuses / penalties]
Longer term QoBiz
Revenue per quarter [banded pricing]
Number of players per month [banded pricing]

Fig. 7. Example QoBiz enabled SLA template

7 Conclusions

The edutain@grid business layer implements a scalable bipartite value chain, underpinned by electronic service level agreements, which can scale gracefully as small ROIAs with low numbers of users grow by several orders of magnitude to large successful ROIAs. We support invoice models that provides variable pricing, banded pricing, cost per duration and penalty fees. These invoice tools provide us with a flexible business layer that supports a variety of incentive mechanisms for key actors in the ROIA value chain.

The edutain@grid project includes two exemplar case study applications; an online multiplayer game and an e-learning multi-student training application. Key quality of service metrics are client packet latency, server frame rate and data throughput. We implement these applications as proof of concept demonstrators. Our business layer implementation supports contract definition using XML service level templates to define pricing and metrics. Contract negotiation follows a discrete offer protocol and contract enforcement is provided by continuous quality of service monitoring, on-demand cross-hoster zone migration and flexible invoice models.

We evaluate our value chains and incentive models in the context of both quality of experience and quality of business. We suggest setting up enhanced incentive models that share rewards between coordinators and hosters, providing a reason for hosters to cooperate with coordinators on cross-hoster load balancing. We show that whilst this is technically achievable the real question to be answered is will an evolving ROIA market see sufficient commercial benefits to make adoption worthwhile.

Acknowledgments

The work described in this paper is supported by the European Union through EC IST Project 034601 'edutain@grid'.

References

1. Andrieux, A., et al.: Web Services Agreement Specification (WS-Agreement), March 17 (2007), `http://www.ogf.org/documents/GFD.107.pdf`
2. Big World Technology (2002), `http://www.bigworldtech.com/`
3. Breuer, D., Erwin, D., Mallmann, D., Menday, R., Romberg, M., Sander, V., Schuller, B., Wieder, P.: Scientific Computing with UNICORE. In: Wolf, D., Münster, G., Kremer, M. (eds.) Procs. of NIC Symposium 2004, Jülich, NIC Series, vol. 20, pp. 429–440. John von Neumann Institute for Computing (2003)
4. Buyya, R., Yeo, C.S., Venugopal, S.: Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: Proceedings of the 10th IEEE HPCC-08. IEEE CS Press, Los Alamitos (2008)
5. Cai, Y.: Networked Gaming: Driving the Future II. Industry report from Parks Associates (2009)
6. Czajkowski, K., Ferguson, D.F., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W.: The WS-Resource Framework (2004)
7. 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)
8. Foster, I., Kesselman, C. (eds.): The Grid2: Blueprint for a New Computing Infrastructure, 2nd edn. Morgan Kaufmann Publishers Inc., Elsevier, Boston (2004)
9. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International Journal Supercomputer Applications 11(2), 115–128 (1997)
10. Helmer, J.: The UK e-Learning Market. Market report commissioned for Learning Light Limited (2007)
11. IBM: Butterfly.net: Powering Next-Generation Gaming with On-Demand Computing (2002)
12. IBM: Google and IBM Announced University Initiative to Address Internet-Scale Computing Challenges (October 8, 2007)
13. Kotsokalis, C.: What's in a Service Level Agreement? SOA@SOI article (2009)
14. Lohr, S.: Google and I.B.M. Join in 'Cloud Computing' Research. NY Times (October 8, 2007)
15. van Moorsel, A.: Metrics for the Internet Age: Quality of Experience and Quality of Business. In: Fifth Performability Workshop, Germany (2001)
16. Pankratius, V., Vossen, G.: Towards E-Learning Grids: Using Grid Computing in Electronic Learning. In: Proc. IEEE Workshop on Knowledge Grid and Grid Intelligence, Canada (2003)
17. Surridge, M., Taylor, S., De Roure, D., Zaluska, E.: Experiences with GRIA — Industrial Applications on a Web Services Grid. In: Proceedings of the First International Conference on e-Science and Grid Computing, pp. 98–105. IEEE Press, Los Alamitos (2005)
18. Vouk, M.A.: Cloud computing—Issues, research and implementations. In: International Conference on Information Technology Interfaces, ITI 2008 (2008)

Monitoring and Fault Tolerance for Real-Time Online Interactive Applications*

Vlad Nae, Radu Prodan, and Thomas Fahringer

Institute of Computer Science, University of Innsbruck,
Technikerstraße 21a, A-6020 Innsbruck, Austria
{vlad,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 a monitoring system we developed which collects data from all the resources in a distributed environment and from the ROIA managed by our platform. We also describe a fault tolerance service which addresses not only the faults commonly encountered in distributed systems, but also faults manifesting at service level, within the platform's management services. Finally, a use-case consisting of the platform running a massively multiplayer online game as a concrete ROIA, is presented in order to demonstrate the roles of the monitoring and fault tolerance services.

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 make them unique in comparison to traditional parameter study or scientific workflows, highly studied by previous Grid research [2]: (1) they often support a very large number of users connecting to a single application instance; (2) 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 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.

Two of the main objectives of the edutain@grid project are unsupervised management 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. In working to achieve these goals in distributed environments, fault tolerant services must be used and dynamic resource and ROIA-session monitoring information needs to be collected and processed. We designed, as part of the edutain@grid management layer, a monitoring service capable of collecting information about the current state and load of resources as well as low-level internal data from ROIA. We also designed and developed a fault-tolerance service which monitors the correct operation of the management services and the employed load distribution and load balancing actions taking the appropriate measures in case of faults.

We introduce the edutain@grid architecture and detail its management layer in Section 2. The monitoring service and the fault tolerance service are described in Section 3 and Section 4, respectively. Section 5 presents a use-case involving the monitoring and fault tolerance services and Section 7 concludes the paper and outlines future work.

2 Architecture

We designed a distributed service-oriented architecture depicted in Figure 1 to support transparent access[1] and scalability for an increased number of end-users (compared to current state-of-the-art) to existing ROIA. A scalable ROIA session is distributed across several ROIA server programs, called *ROIA servers* from here on, that run on distributed resources provided by multiple hosters. A real-time communication framework (RTF) [3] provides the fundamental protocols for parallelising and distributing the ROIA session across multiple ROIA servers. By distributing the load of a session on multiple resources, a larger number of end-users can be accommodated. The architecture is composed of three main actors described in the following subsections.

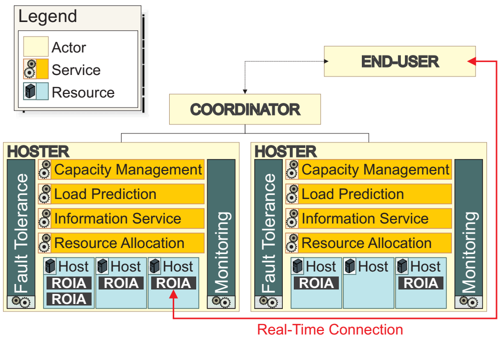


Fig. 1. The edutain@grid architecture. Detail of management services.

2.1 End-User

This actor seeks a connection to a suitable ROIA session, which provides the needed ROIA type and desired QoS. The client ROIA application negotiates

[1] The complex underlying hardware and software stacks are hidden from the end-users.

these terms with the Coordinator, transparently to the end-user. Once the appropriate ROIA session details are obtained from the Coordinator, the end-user connects to the respective ROIA server directly, connection called here "real-time connection", as depicted in Figure 1.

2.2 Coordinator

The coordinator receives from the end-user specific QoS requirements which can be performance-related (e.g. maximum latency, minimum bandwidth, minimum throughput) or ROIA-specific (e.g. ROIA type, number of participants). Its role is to distribute end-users to ROIA servers, which is accomplished as a distributed negotiation between the coordinator and hosters, each of them trying to optimise its own specific metrics expressing individual interests. The relation between the end-user and the coordinator is materialised as a client account and the coordinator-hoster negotiation is finalised as a contract. While the coordinator purely negotiates in terms of end-user-centric QoS parameters, the hosters try to optimise metrics related to their own and often contradicting interests such as maximising resource utilisation or income. Eventually an equilibrium that represents a balance between risks and rewards for all participating parties is reached. The result of the negotiation process is a performance contract that the coordinator offers to the end-user and which does not necessarily match the original QoS request. The end-user has the option to accept the contract and connect to the proposed session, or reject it.

2.3 Hoster

The hoster actor represents an organisation that provides the necessary computational and network infrastructure for running the ROIA sessions, similar to the "resource provider" from the scientific scene. The hoster also runs the management services, all within a *server container*, which monitor the provided resources, steer the hosted ROIA sessions and negotiate new connections with the coordinator.

Resource Allocation Service. Each hoster owns one resource allocation service, responsible for allocating local resources to a large number of connecting end-users. The coordinators make requests based on the load of the ROIA they operate (either statically or dynamically computed), and the hosters respond with offers based on their local time-space renting policy, their internal monitoring data collected by the monitoring service and their load prediction. The resource allocation is realised by a request-offer matchmaking mechanism based on three criteria that favour the hoster [4].

Capacity Management Service. There may occur factors during the execution of a ROIA session which affect the performance, such that the negotiated contracts are difficult or impossible to be further maintained. Typical perturbing factors include external load on unreliable Grid resources, or overloaded ROIA servers due to an unexpected concentration of end-users in certain "hot spots". The capacity management service interacts at runtime with the monitoring service for preserving the negotiated QoS parameters for the entire duration of the ROIA session. Following an event-action paradigm, a violation of a QoS parameter triggers appropriate adaptive steering or load redistribution actions.

Load Prediction Service. The load of a ROIA session depends heavily on internal events such as the number of entities that interact altering each other's state. Alongside internal events, there may also occur external events such as the connected end-user number fluctuation over the day or week with peak hours in the early evening [5]. Hence, it becomes crucial for a hoster to anticipate the future ROIA load.

In our load prediction service, for a fast system reaction time, we employ computationally inexpensive time series prediction models (like exponential smoothing and variants thereof) which generate predictions based on the data collected by the monitoring service. Although their predictive power is limited, they offer sufficiently high accuracy for this kind of trace data and, most importantly, have a very short prediction time which allows the capacity management service enough time to apply its ROIA session steering actions. For massively multiplayer online games, a particular subset of ROIA, we found in [6] a novel algorithm based on neural networks which offers a better accuracy than the aforementioned time series prediction models, while offering the predictions at comparable speeds.

Information Service. To store meta-information about the deployment, invocation, and execution of ROIA, we designed a generic information system with the database schema defined as a composition of independent, generic, type-specific schemas called beans, each bean consisting of one or more customised tables. The information service also stores data generated by the monitoring service and because the ROIA are very dynamic applications and 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 on top of the MySQL database platform. We evaluated this service by carrying out a series of scalability experiments which are detailed in [7].

3 Monitoring Service

The monitoring service's conceptual architecture and its interactions with the involved entities (i.e. monitored entities and clients) are presented in Figure 2.

There are three entities that can be monitored by this service, namely ROIA servers, ROIA sessions and hosts (i.e. the computational and networking infrastructure offered by hosters), for each one, the monitoring service including a profiled probe: ROIA probe, ROIA session probe and host probe, respectively. Each monitoring probe can be attached to a serialization probe whose function is to buffer the monitoring data generated by its attached probe and, eventually, serialize the collected data. Separate probes are constructed for each of the

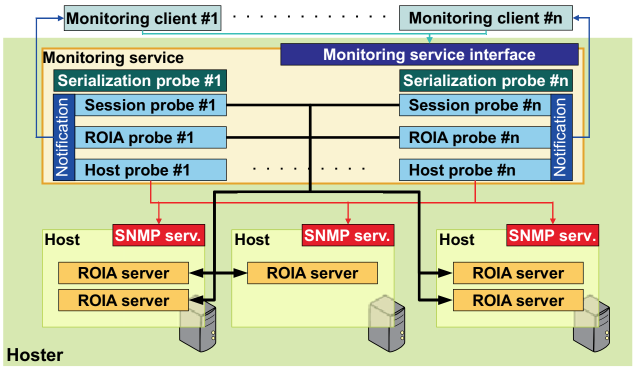


Fig. 2. The architecture of the monitoring service

monitoring service's clients, called *monitoring clients* from here on, and each one represents a separate monitoring session with specific metrics to monitor and monitoring time intervals. If the monitoring clients request notifications for their monitoring, they are registered with the notification dispatcher and each time new monitoring data is generated by their probes, they receive a notification. The monitoring clients are the other management services (all described in Section 2), the coordinator and the end-users through the client ROIA applications or through access portals. Monitoring clients have different privileges which limit the range of metrics they have access to, or the right to serialize data. The management services are granted all the monitoring privileges, whereas the end-users and portals only have access to restricted sets of public metrics.

Figure 3 presents the workflow the monitoring clients must follow in order to utilise the monitoring service:

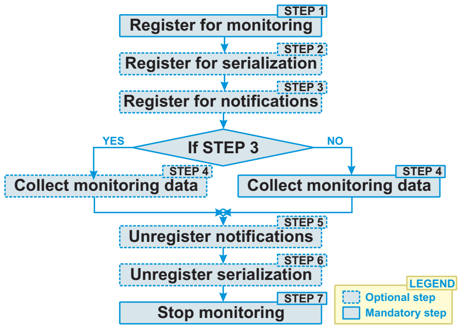


Fig. 3. The monitoring service utilisation workflow

Step 1. Choose the entity to monitor, select the needed metrics to monitor, the monitoring time interval and invoke the monitoring service with these parameters to start the monitoring process. At this point, the monitoring probes are created and started;
Step 2.[optional] Request the serialization of the monitoring data. Now, the serialization probes are created and started;
Step 3. [optional] Request notifications for new monitoring data. Here, the monitoring client is registered with the notification dispatcher;

Step 4. Collect the existing monitoring data. This step has to be performed regularly, but not necessarily at the same time interval as the monitoring is done, thanks to the buffering mechanism provided by the monitoring service;
Step 5. [optional] Stop the notifications for new monitoring data. At this point the monitoring client is unregistered from the notification dispatcher;
Step 6. [optional] Stop the serialization of the monitoring data. Here the monitoring client's serialization probes are destroyed;
Step 7. Stop the monitoring. Here, the monitoring client's probes are destroyed;

Clients can monitor sets of metrics of their choosing at time intervals greater or equal to one second. The monitoring service provides monitoring data buffers for all its probes, whose size can be adjusted prior to the service start. This facilitates the monitoring data collection process for the users because they are thus allowed to collect their data at intervals asynchronous to the monitoring interval. In addition, the monitoring service provides notification callbacks, in case the users need to collect the monitoring data synchronously with the monitoring interval.

4 Fault Tolerance Service

The fault tolerance service is designed to ensure a high level of tolerance to resource faults (to cope with the known problem of highly distributed and heterogeneous systems, like the Grid), as well as to internal faults (e.g. a subset of the management services failing).

4.1 Resource Level Fault Tolerance

In highly distributed and heterogeneous environments like the Grid, a multitude of unexpected events take place which can lead to resource failures, or the impossibility to access or use hosts. The fault tolerance service is designed to cope with faults of the following types:

A. *Host unavailability* The management services continuously check for available hosts and their status in the current setup and will only utilise the hosts which acknowledge and report a functional state;
B. *Host failure* If this event takes place while the host is being used (i.e. there are ROIA servers hosted on it) the management services will change the host's state to "unavailable" and will issue new connection details for the clients serviced by the host in question at the time of the failure. If the event takes place while the host is not involved in any ROIA sessions, then the host is simply reported as "unavailable", and will not be used until it recovers from the fault.
C. *Deployment issue* The management services do not hold exclusive rights to alter deployments. Human intervention, such as non-automated deployments and updates, is allowed which can generate faults. The fault tolerance service does not monitor the deployments (as is the case with hosts), instead they employ a *lazy fault detection* technique which is more efficient as it does not

generate overhead on the monitored hosts nor on the management services themselves. This technique involves the detection of faults in deployments at access time, when the needed ROIA files (e.g. invoking the ROIA executable) and handled accordingly, namely the deployment is marked as "unusable" and the ROIA server is started on other available hosts.

4.2 Service Level Fault Tolerance

As is the case in complex systems, the ROIA management services can be faced with unexpected events, which can lead to service faults. In order to remove or, if not possible, at least minimise the effects of such issues, all management services implement runtime diagnose methods and low-level control interfaces. This enhances the level of control over the management services at runtime, without having to restart the ROIA server container in case of faults or critical problems in any of the services. Moreover, the management services need to function without interruption since they monitor the volatile state of the ROIA sessions; a restart of the management service container is equivalent to a hoster site downtime and additionally the loss of all ongoing ROIA sessions. To this end, we have developed and implemented two monitoring, diagnose and control services designed to watch over all management services and take the necessary countermeasures to prevent and, if necessary, to handle service faults, namely *the service thread manager* and *the service internal state monitor.*

The Service Thread Manager. The management services are loosely coupled and implemented using independent threads, each thread managing a small subset of the functionality of a service. All the threads belonging to management services are automatically registered with the *service thread manager* which transparently monitors their states and activity. All threads can be in only one of the following states at any point during their lifetimes: *sleeping*, *blocked/waiting*, or *working.* The *working* state is not considered safe, thus the *service thread manager* is monitoring the actions of the threads in this state by means of checkpointing. In addition, the *service thread manager* continuously monitors the amount of work (i.e. consumed active processor time) for each thread and, at regular time intervals, assesses their sanity with one of the three defined qualifiers: *safe*, *overloaded* and *hung.* The *sleeping* and *blocked/waiting* states are not prone to faults, thus they are considered *safe* states and they do not require the service thread manager's intervention. Threads spending unusually long amounts of time in the *working* state are marked first as *overloaded,* and eventually, if the problem is not resolved in a predefined amount of time, are marked as *hung.* The *service thread manager* sends signals to all threads at regular intervals in case issues which could impede their normal functionality are detected. All the management service threads implement countermeasures for the *overloaded* and *hung* signals. Each service has its own customised mechanisms to cope with such problems, but the main actions which can be taken when *overloaded* or *hung* signals appear are:

A. For the *overloaded* signal:
  a) If the thread has a variable amount of workload, the sleeping/blocking time or distribution over time of the respective workload can be adjusted (e.g. for a monitoring serialization thread which writes data to disc in uneven chunks, a buffering solution is applied in an attempt to balance the workload between cycles);
  b) If the thread has a fixed workload each cycle, an attempt to share the load with a newly spawned thread can be made, or if by design the thread should not reach this state (e.g. light threads, monitoring threads), it will be flagged as a problematic thread and if the problem persists, its qualifier will be changed to *hung* and the corresponding signal will be sent;
B. For the *hung* signal:
  a.) If the thread receiving this signal does not keep internal data related to the ROIA sessions' states, the *service thread manager* restarts it and resets its associated internal monitoring data;
  b.) If the thread manages data related to the ROIA sessions' states, the *service thread manager* will try to first restart it attempting to reuse its full current internal state, then, on consecutive identical signals, will try restarting the thread and reusing gradually less of its present internal state (where possible) by discarding some of the internal data structures in the reverse order of their importance[2]. If this restart–*hung* cycle continues, the *service thread manager* will eventually restart the thread in question without reusing its internal state. Because this last resort measure can cause some disturbance in the normal activity of the affected service it is implemented only in threads which cannot compromise the global state of the management layer. A relevant example are the monitoring service's threads which are prone to such *hung* signals because of their reliance on network services. They can safely be restarted without preserving their internal states, in the worst case causing a loss of monitoring data on short intervals.

The Service Internal State Monitor. The *service internal state monitor*'s task is to monitor the management services' internal data. This service's purpose is twofold:

- Monitors the internal data generation and implicit memory consumption and prevents faults caused by insufficient memory by issuing *purge* commands. A *purge* command is sent to services as a signal to clear or reduce their data buffers in order to free memory for the new incoming data;
- Provides a safety measure against *memory leaks* (i.e. implementation oversights which cause data that should have been removed from memory to be accidentally kept in memory). The service observes the aberrant memory usage in the faulty threads and orders a cleanup of their internal data. This measure can be destructive (i.e. the service in question could lose important data during the forced cleanup), but it also has the potential to prevent more serious faults which could lead to the service or even service container shutdown.

[2] The importance of a data structure is directly proportional to the likelihood that a fatal failure can occur upon its corruption.

5 Massively Multiplayer Online Game Use Case

We present a use case for the monitoring and fault tolerance services, involving a *massively multiplayer online game* (MMOG), a popular ROIA type, which demonstrates how our ROIA management services cope with a resource-level fault. MMOG are based on a client/server architecture, in which the game server simulates a world via computing and database operations and receives and processes commands from the clients. Based on the actions submitted by the players, the game servers compute the global state of the game world represented by the position and interactions of the entities, and send appropriate real-time responses to the players containing the new relevant state information. The more populated the game world is and the more interactions between entities exist, the higher the load of the underlying game server will be.

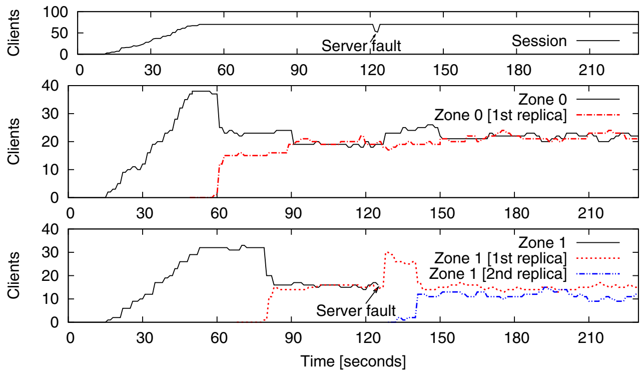


Fig. 4. Monitoring session traces. The clients connected to a game session and their distribution on the game servers within a game session.

We use a MMOG, which supports two methods of load distribution. One, called *zoning*, is based on the spatial partitioning of the game world in zones to be handled independently by separate machines. Clients can freely move between zones by means of transfer portals. The other, called *replication*, distributes load by replicating the same game world (or zone) on several machines [8]. Clients may be transparently migrated between replicated game servers, this representing the actual load sharing mechanism behind this load distribution method. These methods are dynamically employed by our ROIA management services described in Section 2.

Figure 4 shows the total number of clients connected to the distributed MMOG session (top side) and their distribution on the game servers (middle and bottom). The data here shown is collected using the serialization probes of the monitoring service. The session consists of two zones (“zone 0” and “zone 1”), initially running on two machines. At second 15 we initiate a wave of 70 clients connecting to the session, distributed among the two zones. When the initial game servers are getting close to an overload, the management services start replication game servers; for zone 0 at second 60 and for zone 1 at second 78. We simulate a failure of the host running zone 1 at second 125 which results in the disconnection of all the clients connected to it. The fault tolerance service detects the host failure and treats it as described in Section 4.1, by immediately reconnecting the clients to another game server hosting zone 1 (*zone* 1 [1^{st} *replica*]) in order to hide the fault for the involved clients. In a few seconds, because of the additional load generated by this new wave of clients on the *zone* 1 [1^{st} *replica*] server, the capacity management service starts a new replication process and migrates a subset of the zone 1 clients onto it.

6 Related Work

In the area of distributed systems monitoring a significant amount of work has been carried out [9] [10]. While the existing monitoring solutions address distributed systems, some of them having a strong emphasis on performance, they either only support machine and infrastructure level monitoring [9], or only application instrumentation [10]. In contrast, our approach combines the machine level monitoring using the SNMP protocol [11] with the application instrumentation, ensured by the RTF [3], into a single unified system.

Regarding fault tolerance, while comprehensive platforms for fault diagnosis and recovery exist [12] [13] [14], they mostly focus on the correctness of the services' output and root cause of the failures rather than on their overall availability. Our approach lacks the capability to detect the cause of failure in real-time, but, in turn, it has a light design which ensures a fast reaction time in case of failures, thus guaranteeing an almost continuous availability of the managed services which is of utmost importance in ROIA management.

7 Conclusions

We focus our research towards the development of a service oriented ROIA management platform. Here, we presented the architecture and the component services of such a platform. A monitoring system which collects data from resources in the distributed environment as well as from the managed ROIA was described. We also detailed the functionality of our fault tolerance service which addresses not only the resource and deployment related faults, commonly encountered in distributed systems, but also faults manifesting at service level, within the platform's other management services. Finally, a use-case consisting of the platform running a massively multiplayer online game as a concrete ROIA, was presented in order to demonstrate the roles of the monitoring and fault tolerance services.

References

1. Fahringer, T., et al.: 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, p. 530. Springer, Heidelberg (2007)

3. Glinka, F., Ploß, A., Müller-lden, J., Gorlatch, S.: Rtf: a real-time framework for developing scalable multiplayer online games. In: NetGames '07, pp. 81–86. ACM, New York (2007)
4. Nae, V., Iosup, A., Podliping, S., Prodan, R., Epema, D., Fahringer, T.: Efficient management of data center resources for massively multiplayer online games. In: Proceedings of the ACM/IEEE conference on Supercomputing (2008)
5. Feng, W.c., Brandt, D., Saha, D.: A long-term study of a popular mmorpg. In: NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, pp. 19–24. ACM, New York (2007)
6. Nae, V., Prodan, R., Fahringer, T: Neural network-based load prediction for highly dynamic distributed online games. In: Proceedings of 14th International Euro-Par Conference, pp. 202–211 (2008) ISBN 978-3-540-85450-0
7. Nae, V., Herbert, J., Prodan, R., Fahringer, T.: An information system for real-time online interactive applications. In: Euro-Par 2008 Workshops, pp. 352–361. Springer, Heidelberg (2009)
8. Müller, J., Gorlatch, S.: GSM: a game scalability model for multiplayer real-time games. In: Infocom. IEEE Computer Society Press, Los Alamitos (2005)
9. Newman, H.B., Legrand, I., Galvez, P., Voicu, R., Cirstoiu, C.: Monalisa: A distributed monitoring service architecture. CoRR cs.DC/0306096 (2003)
10. Gunters, D., et al.: Dynamic monitoring of high-performance distributed applications. High-Performance Distributed Computing 0, 163 (2002)
11. Case, J., Fedor, M., Schoffstall, M., Davin, J.: A simple network management protocol (snmp). rfc 1157. Technical report, Network Working Group (1990)
12. Abd-El-Malek, et al: Fault-scalable byzantine fault-tolerant services. In: SOSP '05, pp. 59–74. ACM, New York (2005)
13. Hofer, J., Fahringer, T.: Grid application fault diagnosis using wrapper services and machine learning. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 233–244. Springer, Heidelberg (2007)
14. Dialani, V., et al.: Transparent fault tolerance for web services based architectures book. In: Monien, B., Feldmann, R.L. (eds.) Euro-Par 2002. LNCS, vol. 2400, pp. 107–201. Springer, Heidelberg (2002)

A Service-Oriented Interface for Highly Interactive Distributed Applications

Frank Glinka, Allaithy Raed, Sergei Gorlatch, and Alexander Ploss

University of Münster, Germany
{glinkaf,a.raed,gorlatch,a.ploss}@uni-muenster.de

Abstract. The emerging class of Real-time Online Interactive Applications (ROIA) include massively-multiplayer online games and e-learning applications. They pose completely new challenges for application developers, including very high level of user interactivity with real-time QoS requirements on distributed performance and scalability. We describe a service-oriented interface that comprises: (1) the Real-Time Framework (RTF) supports a high-level application development process which frees the software developer from the low-level details of distributed computation and communication; (2) the Hoster Management Interface (HMI) supports the transparent resource management for a running application, in particular the creation, controlling and monitoring of ROIA instances. We present our efficient implementation of the interface and describe its use for two particular distributed application scenarios.

1 Introduction

This paper is motivated by an emerging class of Internet-based applications – *Real-Time Online Interactive Applications* (ROIA). Popular and market-relevant representatives of ROIA are multiplayer online computer games, as well as training and e-learning applications based on high-performance simulation.

The challenging features of ROIA which distinguish them from traditional distributed applications are as follows:

- *Highly intensive user interaction*: the application supports a very large number of users which interact with each other (many-to-many) very often and in a real-time manner.
- *Concurrency at a single application instance*: several users connect to a single instance, thus sharing the application, cooperatively or competitively.
- *Ad-hoc connections*: users connect to applications in an ad-hoc manner, at times of their choice, and register anonymously or with different pseudonyms.
- *High Quality of Service (QoS)*: the applications must maintain certain strict QoS parameters related to the user interactivity (response time, etc.).

We aim at using the concepts of grid and service-oriented computing for overcoming the main problems of ROIA: cumbersome low-level programming, manual hosting, static resource management, and no Quality of Service (QoS) guarantees, which become problematic with a massive number of simultaneously active users. Efficient development tools and resource management for ROIA will allow these applications to become a "killer application" for the Internet of Services and Grid computing, and increase the visibility of these technologies by targeting a huge community of non-expert users who are ready to subscribe and pay for provided services.

In this paper, we present a novel, service-oriented interface for developing and running ROIA that consists of two parts addressing two main challenges:

- The Real-Time Framework (RTF) simplifies the development of scalable grid-enabled applications. The framework implements various parallelization concepts and introduces integrated monitoring and controlling facilities. Applications on top of this framework are developed as a collection of services.
- The *Hoster Management Interface* (HMI) connects the development of scalable, grid-enabled real-time applications and the on-demand resource management for their operation in a service-oriented approach. We show how this is connected to the corresponding business needs.

We briefly describe a highly optimized C++ implementation of the interface and present its use for two kinds of multiplayer online games. As compared to our previous work [1,2,3], this paper introduces the new design of HMI and its integration with RTF, and demonstrates the usage for developers and service providers. The integration allows an application developer to focus on application-specific parts when working with RTF and not being bothered with service provision related aspects (e.g., authentication and security issues, resource management) while the HMI is used by the application service provider for the run-time resource management or setup of security.

The paper is organized as follows. Section 2 describes the multi-layered architecture of edutain@grid [4,5], and Section 3 then focuses on the application development within the real-time layer. The service-oriented interface that interconnects real-time and management layer is presented in Section 4 and followed by the discussion of the implemented prototype and the available demonstrator applications in Section 5. A summary and discussion of related work and our future studies concludes the paper in Section 6.

2 The Multi-layered Architecture of ROIA

The main reasons that still hinder the development of efficient multiuser applications for the future Internet are their real-time and business-related requirements and the complexity of making applications scalable and grid-enabled. Multiplayer online games, e.g., require fast and low-overhead communication facilities with extremely short response time to client's inputs (less than 100ms). On the business side, besides security, *service level agreements* (SLA) are required based on *quality of service* (QoS) related metrics of real-time applications, as clients continuously expect a certain QoS while using a real-time application. Examples for such metrics are the response time or the possibility to join or leave an application instance at any time. Shortcomings of the current grid technology are:

(1) slow, often web-service based communication facilities which are not suitable to drive real-time computation and communication, and (2) the missing support for business models and on-demand resource management that are mandatory for commercial applications in this domain.

To reduce the complexity of the application development and operation, we develop a multi-layered architecture that separates all the related aspects of the application development and operation into three major categories:

– The *real-time aspects* cover the development of applications that rely on fast and grid-aware communication facilities and sophisticated parallelization support. Monitoring and controlling facilities are also related to these aspects, although they are of particular interest for resource management.
– The *management aspects* cover application controlling and monitoring, as well as resource allocation to the grid resources. Also the transition of SLA and QoS to the necessary resource management is covered.
– The *business aspects* cover the payment-, security- and QoS-related topics relevant for the application service provisioning and the client access to it.

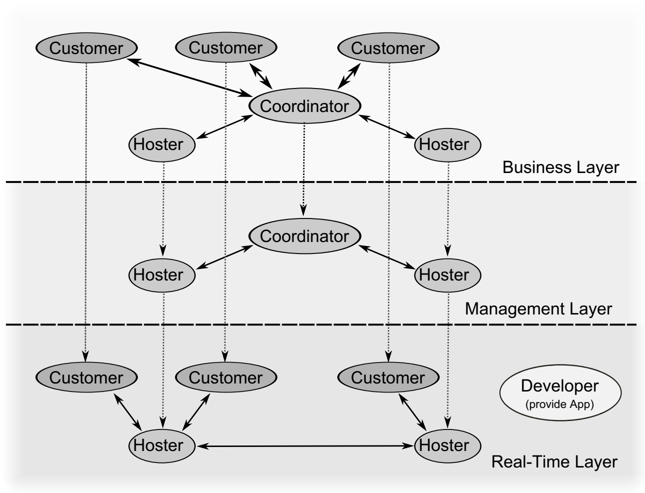


Fig. 1. ROIA layered architecture

Each layer of our multi-layered architecture deals with a particular aspect, although not all aspects can be exclusively assigned to one layer only. Figure 1 illustrates the three corresponding layers from business (top) over management to the real-time layer (bottom). This architecture represents the natural transition from high-level business-related agreements between application service providers and customers to the low-level operation of the real-time applications.

The edutain@grid project analyzed the business actors, value chains and workflows that are present in the current market models and have to be supported by the presented architecture [1]. The four major actors involved are:

- *Application developer*: develops the application and its multimedia contents.
- *Hoster*: an organization that provides computational and network infrastructure and hosts ROIA deployments and services.
- *Coordinator*: an organization that makes a ROIA accessible to customers and coordinates one or more hosters to deliver the required application service.
- *Customer*: accesses the ROIA via the coordinator. Customers are, e.g., online game players or e-learning instructors and students.

Figure 1 illustrates the four actors within the multi-layered architecture. An SLA between hoster and coordinator, e.g., can specify that a hoster is obliged to operate application X for up to hundred concurrent users while maintaining a server response time of at most 100 ms for a fixed payment. A customer, on the other side, has a contract with the coordinator that guarantees the access to an instance of application X for an agreed price. This is one of the variety of use cases supported by the multi-layered architecture.

In Figure 1, links across layer boundaries are only present within one organization. For example, a hoster in the business layer is connected with itself in the management layer. By avoiding cross-layer interactions, each organization can manage actions and respond at the appropriate level, passing instructions down to the layers below, and feeding back information and exceptions to the upper layers. The flow of information from top to bottom urges different components to fulfill the hoster's commitments to other organizations. The flow from bottom to top keeps the management layer informed of any failure or exhaustion of critical resources, and keeps the business layer informed about the available capacity, thus ensuring that the hoster does not offer new SLA terms which cannot be fulfilled. Each actor controls its own resources and interacts with other actors to fulfill the SLA. There are no global information services or decision point, which enables high scalability to support thousands of different actors.

3 ROIA Development: The Real-Time Layer

In order to enable the management- and business layer to realize the application service provisioning, a grid-aware ROIA must implement a variety of management-related functionalities: start or stop application, join or remove customers, monitor QoS, add or remove resources, etc. Although these functionalities are related to the service provisioning, the developer's expertise is usually restricted to the technical implementation within the corresponding application context. Our goal is that all the business- and management-related aspects of forming SLA, different types of SLA and the resource management should be hidden from the developer in the real-time layer.

To realize this separation of concerns, we provide the highly optimized C++ *Real-Time Framework* (RTF) to the developer which provides the following features a)-e).

a) Simplified development of scalable and grid-enabled ROIA. The framework liberates the developer from the low-level communication-related tasks and supports three different parallelization concepts for ROIA: zoning, instancing and replication. The high-level development methodology which we offer together with RTF [2] abstracts from the underlying (grid-)resources, such that developers can focus on how to realize their application accordingly to the chosen parallelization concept. This development approach simplifies the adoption of grid computing for ROIA.

b) Integrated monitoring and controlling facilities. To allow management- and business layer to monitor and control ROIA service, we ensure that the run-time characteristics are exposed and controlling commands are accepted by a ROIA. This allows, e.g., the management-layer to monitor the performance of an application and assign new resources if required. RTF can automatically expose a configurable amount of information: communication latencies and bandwidth consumption, number of processed client messages, CPU load, etc.

c) Applications developed as a collection of services. Besides the communication and parallelization aspects of the ROIA development, also persistence, audio chat, authorization management may be relevant for the application. RTF provides additional services that provide easy-to-use audio communication and persistence services to be usable within an application.

d) Decoupling the real-time layer from the management- and business layers. While events and update processing within the ROIA are only allowed to take milliseconds, the procedures within the management- and business layer may take multiple seconds. For example, an SLA between a coordinator and a hoster might be negotiated in multiple steps, potentially including human decision-makers, until it is finally formed and a hoster's action of resource addition to a ROIA might take seconds to be finished.

e) Translation between C++-based real-time applications and the Java- or web-services-based management and business layers. The technologies that are used within the different layers differ. While ROIA are traditionally realized in the game industry using C++ in order to allow real-time-aware and highly optimized implementations, the management and business layer are usually Java- and web-services based.

The features a)-e) of RTF allow the ROIA developer to concentrate on the application development within its domain of knowledge. Moreover, the access to the application is automatically provided by the RTF to the management and business layer. Therefore, the upper layers can create and operate ROIA instances and can manage them during run-time to deliver a certain QoS promised to coordinators or customers.

RTF is realized in a modular-based manner, each module providing a specific set of services to the application developer and the upper management- and business layers. Figure 2 shows the modules of RTF and the connection to the customers and upper layers. The developer can choose which particular module to use in its application, it is not forced to use all of them. But if the controlling and monitoring module is not used, then no dynamic and extended management options are available and the upper layers are only able to start/stop/kill ROIA instances. Figure 2 also shows a connection between the controlling and moni-

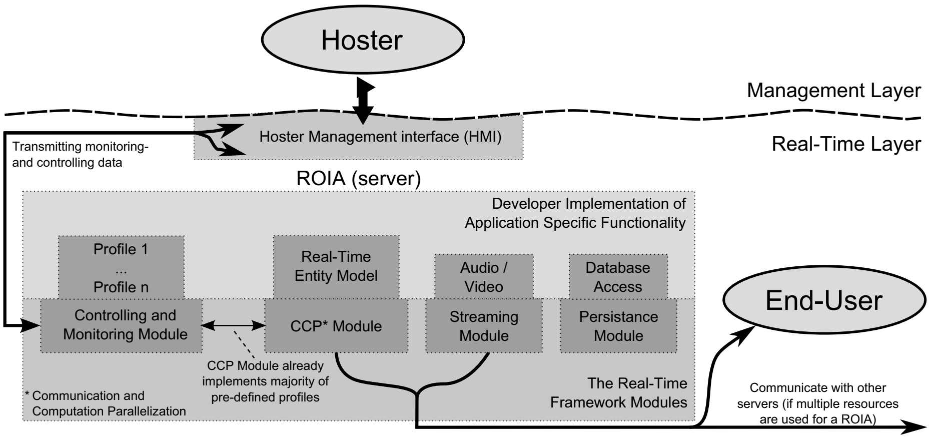

Hoster
Management Layer
Real-Time Layer
Hoster Management interface (HMI)
Transmitting monitoring- and controlling data
ROIA (server)
Developer Implementation of Application Specific Functionality
Profile 1 ... Profile n
Real-Time Entity Model
Audio / Video
Database Access
End-User
Controlling and Monitoring Module
CCP* Module
Streaming Module
Persistance Module
CCP Module already implements majority of pre-defined profiles
* Communication and Computation Parallelization
The Real-Time Framework Modules
Communicate with other servers (if multiple resources are used for a ROIA)

Fig. 2. The RTF Modules and the connection to the management layer

toring module and the *Communication and Computation Parallelization Module* (CCP Module) of RTF because the CCP Module implements a lot of the standard controlling/monitoring profiles for the developer. The CCP Module is the RTF component which realizes the distribution and communication functionality of a ROIA for the developer. The module can gather a lot of relevant information for the monitoring profiles as it is well integrated with the application state. More information on profiles follows in Section 4.

4 The Hoster Management Interface (HMI)

ROIA which are implemented on top of RTF are offered as services that will be deployed and operated in practice by a hoster (organization hosting games). A hoster that chooses to offer a ROIA as a service to the coordinator or a customer first deploys a ROIA on its resources and then integrates it into its resource management system.

To create, control and monitor a ROIA instance, we develop the service-oriented *Hoster Management Interface* (HMI). Figure 3 shows the two components of the HMI:

1. A *ROIASessionManager* instance is the access point for the hoster to the management of ROIA. A ROIA session represents an application instance which is composed of one or several participating ROIA processes. A ROIA process represents a system process on a particular resource of the hoster that is executing a single-server or parts of a distributed multiserver application. Besides the ROIA creation, the ROIASessionManager allows a hoster to monitor and control the run-time behavior of the application through the monitoring and controlling profiles implemented by the application.

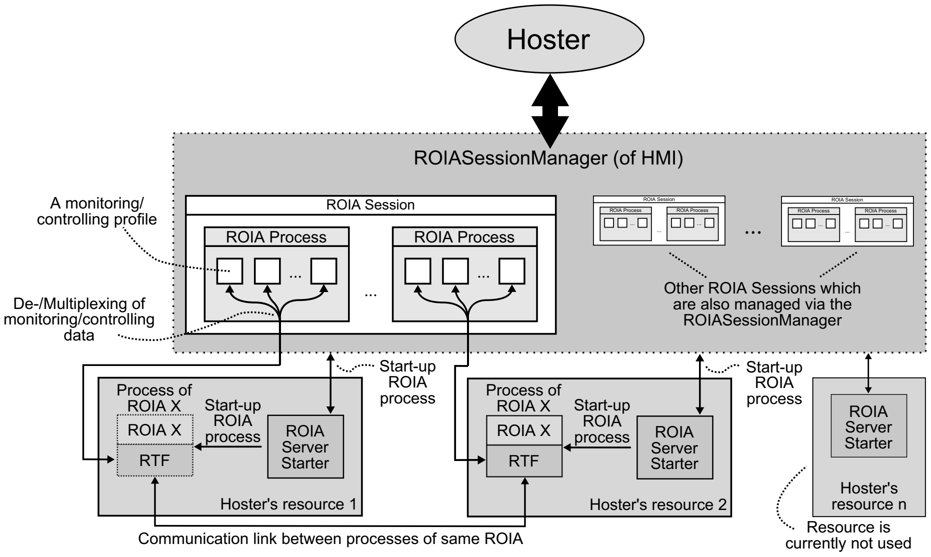

Hoster
ROIASessionManager (of HMI)
ROIA Session
ROIA Process
ROIA Process
A monitoring/ controlling profile
De-/Multiplexing of monitoring/controlling data
Other ROIA Sessions which are also managed via the ROIASessionManager
Start-up ROIA process
Process of ROIA X
ROIA X
RTF
ROIA Server Starter
Hoster's resource 1
Hoster's resource 2
Hoster's resource n
Resource is currently not used
Communication link between processes of same ROIA

Fig. 3. The hoster management interface to the ROIA

2. The ROIAServerStarter is a light-weight service that must be deployed by the hoster on each of the available resources. This service is used by the ROIASessionManager to start a ROIA on a particular resource.

4.1 Setting Up a ROIA Service via HMI

The start-up of a ROIA service is triggered by the coordinator which requests the availability of a specific ROIA (e.g., for a new customer) from the hoster. This availability is assured by the hoster to the coordinator by previously formed SLA. Upon the coordinator's request, the hoster conducts the following steps:

1. A ROIA session is created via the ROIASessionManager which initially contains no ROIA processes and consumes no resources. The ROIASession-Manager provides from now on an access to this ROIA session which, e.g., contains the application name, version and a list of the participating ROIA processes (none at the beginning).
2. A ROIA process is created via the ROIASessionManager on one of the hoster's resources. The ROIASessionManager uses the ROIAServerStarter on the particular resource to start the corresponding system process. Although the ROIA process is active on the resource - it does not process any application data so far.
3. The hoster now can add new zones, instances or replicated areas to a process. This step is application-specific and the hoster needs to have the appropriate meta-data about the application which specifies which zone/instances/replicated areas can or must be started.

Table 1. Monitoring Profiles provided by RTF

Module	Purpose
RTFNetwork	Gives detailed information regarding communication bandwidth and latency of the communication. Most of this data is provided internally by RTF itself.
RTFClient	The application can report client-specific status information via this interface.
RTFEntityModel	If the CCP module is used, this profile reports detailed information about the number and location of dynamic entities or the segmentation of the virtual environment. Most of the data will be provided internally by the CCP module itself.
RTFRealTimeApplication	Reports generic monitoring data of ROIA, especially the performance metrics like saturation of the real-time loop and the load of the application process.
RTFSystem	Provides information about the uptime of the application, the number and addresses of other connected servers, etc.
MonitoringPassThrough	A generic profile; allows the application to report arbitrary <key,value> data.

After the setup is completed and the ROIA service is operating, the hoster can monitor and control the ROIA through the monitoring and controlling profiles that are implemented by the ROIA.

4.2 Controlling and Monitoring via HMI

The controlling part of the controlling and monitoring module provides the possibility to send controlling commands directly into a running ROIA process. Such commands may be very different for different applications: For example, a multiplayer action game may incorporate the possibility to change the game world environment (usually called map) during runtime, while an e-learning application does not incorporate such maps and, therefore, does not support a corresponding command. However, an e-learning application may require other application-dependent commands, like switching into a more secured test mode or loading the next lesson. For supporting different commands for different ROIA, the controlling and monitoring module offers various profiles to be implemented by the particular application. For a particular profile, RTF offers a C++-interface to be implemented by the application developer. Currently, three controlling profiles have been specified: a generic ROIA profile and two dedicated profiles for online games and e-learning applications, respectively. For an additional class of ROIA to be supported by the edutain@grid system, new profiles can be specified by the developer itself and used within the controlling and monitoring module.

The real-time layer receives the particular controlling commands to be sent to a particular ROIAProcess from the management layer and transfer the command to the RTF instance running inside the process. The controlling module inside that RTF instance forwards the command to the particular target profile implementation.

Monitoring constitutes the counterpart to the controlling part: here, internal application-specific as well as ROIA-general status information is now sent from the ROIAProcess out to the edutain@grid system. Here, again, different values of interest are organised in different profiles, which the application has to implement in order to supply the corresponding monitoring information. The thus obtained monitoring data is made available to the upper business and management layers via the real-time layer interfaces.

The various monitoring data to be supplied by the applications is organised in different profiles, of which the application can implement the suitable ones. The following table provides an overview of the so far specified profiles (see Table 1). RTF helps in acquiring data relevant for these profiles and provides the pass-through functionality to the upper business and management layers.

5 Experimental Evaluation

We have evaluated the presented service-oriented architecture using two applications from the class of multiplayer online computer games. The main objective of the evaluation is to verify the architectural concepts, estimate the quality of the developed services for different groups of actors and measure different performance characteristics, in particular scalability.

Our first designated application is a future commercial product which is currently being developed as a project demonstrator within the edutain@grid project to evaluate and exploit the project achievements. The French game developer studio *Darkworks S.A.* [6] develops a fast-paced action game.

Figure 4(a) shows a screenshot of the current prototype which already uses our described RTF, together with the service-oriented HMI interface. The contents of the game are as follows: Multiple players are assigned to teams and try to capture hovering spheres while others try to pre-empt or disturb the capturing. RTF allows the game developers to concentrate on the implementation of the game logic and GUI. They never get in touch with SLA-related topics like forming a legal SLA, different kinds of SLA or SLA breaches. We get very positive feedback from the company about the RTF API and RTF's replication functionality, as well as the automatic monitoring; all these features are already used. The resource management services which are currently supported include the start-up, shutdown as well as the seamless run-time migration and resizing of a game session using multiple resources.

Our second test application – Bioclysm – is shown in Figure 4(b). Bioclysm is a massively multiplayer online role-playing game which was designed and implemented at the University of Muenster and exploits the whole variety of the parallelization concepts offered by RTF. Clients move their avatars though a 3D environment, interact and attack other participants or contact other elements of the game world.

(a) Darkworks' demonstrator (b) Bioclysm demonstrator

Fig. 4. Two applications based on RTF exploiting HMI's functionalities

We used the Bioclysm application to test the described HMI-based resource management between management- and real-time layer for the following practice-relevant scenario. Multiple customers have a contract with the coordinator that guarantees the access to an instance of the game which supports 500 simultaneous players and should maintain a minimal response time of 100 ms. As soon as the coordinator has selected the hoster of its choice (maybe the cheapest one or with highest promised QoS), it requests a session of Bioclysm under the constrains of the contracts with its customers (response time and supported number of simultaneous players). The hoster matches these requirements against the available resources and starts the session via the HMI on as little resources as possible. When players start to join the Bioclysm session, the hoster dynamically adds resources, in order to accommodate up to 500 players with <100 ms response time. If a considerable amount of players leaves the game then underutilized resources will be dynamically removed from the Bioclysm session. The usage of all concepts offered by RTF enables Bioclysm to be scalable for high client numbers during one single game session; at the same time the game session is automatically controlled and monitored.

Finally, our experiments addressed the overhead introduced to the runtime of ROIA by the high-level RTF framework. The question is how large is the price to be paid for the additional comfort offered to the application developer. For our study, we used a very popular multiplayer online game Quake 3 which belongs to the most challenging genre of First-Person Shooter games. It is known to be one of the best optimized games regarding responsiveness and computational load. We developed a port of the open source Quake 3 version from a single- to a multiserver implementation using the RTF middleware described in this paper.

Our preliminary results already show that the ported version achieves an equal responsiveness compared to the original Quake 3 while the CPU utilization increases from about 20% with the original Quake 3 to 40% with the RTF version with 24 players, see [3] for details. Future work will start to optimize RTF regarding the computational overhead in order to reduce the gap between the original Quake 3 and the RTF version.

6 Related Work and Conclusion

We described a service-oriented approach to the development and run-time support of the novel class of Real-Time Online Interactive Applications (ROIA) that allows to develop these distributed challenging applications at a high level of abstraction and to organize a dynamic resource management at run-time.

A work related to ours is performed within the BEinGrid project which conducts a business experiment [7] for a virtual hosting environment for distributed online games which investigates how application service providers can rapidly deploy and manage their services in a secure and accountable way. Also [8] discusses a hosting environment which is able to start and stop single instances of a game on-demand on a given resource pool. Both infrastructures support the hosting of games and real-time applications, but little is said so far about how these applications, used by multiple customers simultaneously, adapt to changing resource demands. In contrast, the aspect of the dynamic resource management is the focus of our HMI interface together with the RTF middleware.

Frameworks which investigate the dynamic runtime composition of services under hard real-time constraints are, e.g, investigated in [9,10]. Both frameworks work with worst-case execution time throughout the system and discuss how scheduling and the overall architecture is affected by the real-time constraints (e.g. communication performance, discovery, etc.). However, ROIA are not always decomposable into separated services and barely have an assessable worst-case execution time. Therefore, we have described in this paper special mechanisms for a ROIA resource adaption at the hoster (service provider) side, which allow the hoster to assign the appropriate amount of resources necessary to fulfil its service commitment.

We evaluated our approach using two real-life case studies. The HMI interface, which is the main contribution of this paper, has allowed in both games to go far beyond the current state of the art in steering functionalities for multiplayer online games: a game is solely started and stopped with a predefined resource allocation setup. HMI adds integrated monitoring and controlling facilities and allows to add resources to a game if the current user demand requires this, either because a lot of players joined or left the game or because they use more compute-intensive parts of the game. The main advantage of our approach is that the online game is now provided as an adaptable service that adjusts to the actual service demand.

Besides the in-depth evaluation of RTF's performance characteristics, we are currently investigating how the presented HMI can be integrated with infrastructure services like Amazon's EC2 [11]. The resources of such a service can be easily used for the ROIA service provision as a ROIA can be deployed into a virtual machine package without any problems. This way, a dedicated hoster could even use external resources for, e.g., an unexpected resource shortage. Although the hoster can outsource some of its hardware requirements this way, it still covers the risk of maintaining the ROIA service under the terms of an SLA with the coordinator.

On the application side, we are investigating the applicability of our approach to a broader class of applications. RTF is currently integrated into an interactive marine safety e-learning application [12], as well as into a real-time interactive crowd simulation which uses RTF's distribution and scalability features.

References

1. Ferris, J., Surridge, M., Watkins, E.R.: Business value chains in real-time on-line interactive applications. In: Altmann, J., Neumann, D., Fahringer, T. (eds.) GECON 2008. LNCS, vol. 5206, pp. 1–12. Springer, Heidelberg (2008)
2. Glinka, F., Ploss, A., Gorlatch, S., Müller-Iden, J.: High-level development of multiserver online games. International Journal of Computer Games Technology 2008(5), 1–16 (2008)
3. 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 '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, Yokohama, Japan, December 2008, pp. 83–90. ACM, New York (2008)
4. The edutain@grid project (2009), `http://www.edutaingrid.eu`
5. Fahringer, T., Anthes, C., Arragon, A., Lipaj, A., Müller-Iden, J., Rawlings, C.J., 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)
6. Darkworks s.a. (2009), `www.darkworks.com`
7. BEinGRID - Distributed Online Gaming (2009), `http://www.beingrid.eu/be9.html`
8. Shaikh, A., Sahu, S., Rosu, M., Shea, M., Saha, D.: Implementation of a service platform for online games. In: NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, pp. 106–110. ACM, New York (2004)
9. Tsai, W.T., Lee, Y.H., Cao, Z., Chen, Y., Xiao, B.: Rtsoa: Real-time service-oriented architecture. In: SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering, Washington, DC, USA, pp. 49–56. IEEE Computer Society, Los Alamitos (2006)
10. Estevez-Ayres, I., Almeida, L., Garcia-Valls, M., Basanta-Val, P.: An architecture to support dynamic service composition in distributed real-time systems. In: ISORC '07: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, Washington, DC, USA, pp. 249–256. IEEE Computer Society Press, Los Alamitos (2007)
11. Amazon EC2. Amazon Elastic Compute Cloud (2009), `http://aws.amazon.com/ec2`
12. Gorlatch, S., Glinka, F., Roreger, H., Rawlings, C.: Distributed e-Learning using the RTF middleware. In: Proceedings of the 2nd Annual Forum on e-Learning Excellence in the Middle East, Dubai, UAE, January 2009, pp. 239–255 (2009) ISBN 978-9948-15-026-8

CompTalks – From a Meta-model Towards a Framework for Application-Level Interaction Protocols

Eryk Ciepiela, Maciej Malawski, and Marian Bubak

Academic Computer Centre CYFRONET AGH, Nawojki 11, 30-950 Kraków, Poland
Institute of Computer Science AGH, Mickiewicza 30, 30-059 Kraków, Poland
eryk.ciepiela@cyfronet.pl

Abstract. This work presents CompTalks – a novel concept and meta-model for specifying application-level communication protocols. The goal is to enable custom fine-grained and elaborate message exchange between distributed yet tightly-coupled parties. Hence, the concept of a conversation protocol is introduced. Its reference implementation – the CompTalks Framework – is a Java-based middleware toolkit that supports development, testing, analysis, validation and running highly interactive services. An important feature is the ability to verify the developed protocols at compile time by using a Petri Net-based analyzer. The framework was successfully applied to develop a protocol for GSEngine which serves as the runtime system of the ViroLab virtual laboratory, enabling development and execution of complex collaborative applications.

Keywords: Service-Oriented Architecture, interaction, conversation, application-level protocols, application framework, Java, Petri Net.

1 Introduction

The subject of the research discussed in this paper is focused on paradigms and implementation aspects of stateful and highly interactive services in Service-Oriented Architectures. Our work was motivated by the need for development of a new protocol between the GSEngine [1] client and server which constitute the core of the runtime system of the ViroLab virtual laboratory [2, 3]. The requirements included interactive and collaborative execution of complex applications (experiments) on the server together with online streaming of input and output data together with user interaction message exchange with the client. The rationale for our research is that a whole class of stateful and interactive services is not adequately addressed by the state-of-the-art paradigms and technologies applied in modern service-oriented distributed systems.

Currently available service models and frameworks assume the services are loosely coupled with clients and that their interfaces comprise a set of advisably idempotent operations which are to be invoked in a blocking and synchronous manner. Message-Oriented Middleware (MOM) enables asynchronous messaging, however it only offers predefined styles of messaging (e.g. queues and publish-and-subscribe topics) and is therefore intended for loosely-coupled parties. In both approaches consecutive invocations of operations or message passing proceed in no implicit session context, which leaves system designers with few options to explicitly preserve such context. As a result, session tracking code often becomes mixed with service business logic code, preventing it from being system architecture-agnostic and thus reducing its reusability and maintainability.

Current approaches do not suffice for use cases that need finer-grained interaction when:

- Input data is provided to the service in parts – e.g. in a continuous way, in batches, periodically etc.
- The service asks its client to make decisions that determine further processing – e.g. the service performs some steps, but there are check points when it has to ask a decision-maker for further processing instructions, validation, confirmation etc.
- Input data is heavyweight and is the subject of a chain of service operation invocations which are not known beforehand – e.g. input data is sent once, before the client decides which processing chain to apply to this data.
- The service has to ensure richer client experience and responsiveness – e.g. inform about processing status, provide the client with intermediary and partial results and other relevant information from the client's point of view.

In order to address these issues, we propose a new concept of a Conversation Protocol as a way for modelling interaction between services and their clients that is conceptually simplistic, yet generic and scalable enough to be capable of describing complex stateful interactions schemes. Its goal is to natively support session context preservation, asynchronous communication, highly interactive data and control flows. The proposed Conversation Protocol Meta-Model further allows for formal modelling of conversation protocols, thus enabling analysis, simulation and validation e.g. by using the Petri Nets [15] model. The above mentioned concepts are implemented in our Java-based CompTalks Framework which serves as a basis for the GSEngine protocol.

The paper is organized as follows: Section 2 discusses the state-of-the-art architectures and technologies. The Conversation Protocol concept is introduced in Section 3 with further details including formal modelling of conversation protocols enabling analysis, simulation and validation that use the Petri Nets model (Section 4). Section 5 the presents Java-based CompTalks Framework which is a reference implementation supporting Conversation Protocols. Performance tests are reported in Section 6. We conclude with the evaluation of the advantages and limitations of the proposed solution and an outline of the future prospects for CompTalks in Section 7.

2 State of the Art

The Web Services [4] approach considers a service interface as a set of operations which typically are to be invoked sequentially in a blocking, synchronous and asymmetric request-response manner. Another shortage of the Web Services Description Language (WSDL) [5] is that it does not cover the intended sequence in which operations need to be invoked in order to carry out given use case scenarios.

Such directions are not formalized, forcing developers to refer to additional, usually plaintext, documentation, which may lead to improper use of services.

The Web Services Conversation Language (WSCL) [6] proposes a specification for describing the flow of documents exchanged between a Web Service and its client. Such descriptions are intended to be interpreted by Web Service infrastructures and development tools. Although the specification enables describing conversation protocols, it is dedicated to loosely-coupled parties interacting via document exchange. Hence, it is focused on building another abstraction layer over services communicating with SOAP-like [7] protocols rather than on communication protocols allowing for finer-grained interaction and tighter coupling of parties.

Some Web Service-oriented approaches support operations on the state associated with a service. In such cases, the result of an operation depends on prior operations. Web Services Resources (WS-R) [8], Web Services Transfer (WS-T) [9] and Representational State Transfer (REST) [10] follow similar paradigms [11] for managing state and offer advanced access mechanisms, e.g. notifications about state changes or accessing the state in parts. Such an approach can be regarded as state model-centric and document-based; therefore it is poorly suited for interaction-oriented applications, which require a means for specifying custom interaction schemes that reach beyond those offered by the aforementioned specifications.

The Extensible Messaging and Presence Protocol (XMPP) [12] is an XML-based protocol for near-real-time messaging, presence, and request-response services [13]. Contrary to WSCL it enables finer-grained interaction and bidirectional stream-like communication. It originally served as a streaming medium for Instant Messaging, the message exchange cannot be managed by any protocol rules thus cannot support custom application-level conversation protocols.

The Blocks Extensible Exchange Protocol (BEEP) [14] is an attempt to provide a generic application-level meta-protocol for connection-oriented, asynchronous interactions. It enables defining various styles of message exchange (request-response, asynchronous calls and streaming) but keeps it asymmetric with only the server being capable of streaming and generating responses and the client limited to sending requests.

In order to enable interactive connectivity to grid infrastructure nodes several utilities like *glogin* [21] emerged that patch grid toolkits in order to maintain direct client-service bidirectional data streaming channel. This constitutes powerful foundation for interactive message exchange but still needs high-level model for specifying application-level protocols.

3 CompTalks Conversation Protocol Concept

The CompTalks Conversation Protocol proposes a novel and more general way for defining and describing service interfaces. In CompTalks, an interface to a service is regarded as an interaction protocol that specifies messages and rules of message passing between a service and its clients. More precisely, the description of an interface consists not only of a set of messages the service can exchange, but also contains information about the allowed sequences of messages sent between a service and its client. Allowed sequences are specified through a state machine with transitions denoting messages sent to or from a service. Interactions between endpoints in CompTalks reach beyond the asymmetric client-server request-response model by supporting stateful, asynchronous, non-blocking and symmetric communication in order to enable elaborate, interactive and finer-grained message exchange schemes.

In CompTalks a *conversation* is defined as a message exchange that follows some rules that are agreed a priori by communication endpoints. We distinguish *basic conversation rules* that constitute a rudimentary contract between communicating endpoints, guaranteeing effectiveness, predictability and determinism of conversation. Over that, application-specific rules, expressed in the form of a *conversation protocol,* define allowed messages and message sequencing.

Basic CompTalks conversation rules are codified as follows:

1. They assume a *medium* for carrying messages comprises two streams – one for each direction between the client and the server. Messages are delivered in the order in which they were inserted into the stream. Latency and jitter in message delivery is allowed. Depending on use case reliability has or has not to be ensured. These requirements (including reliability) are met e.g. by the Transmission Control Protocol (TCP) – therefore, the Internet TCP/IP protocol stack can be successfully used as a medium.
2. On each side of a medium there is exactly one *endpoint*. Each endpoint specifies the *messages* it can receive. A message, in turn, specifies in which *conversation state* it can be sent, which conversation state ensues after it is sent and whether it hands over *control* over the conversation. Control denotes whether the endpoint, after receiving such a message, is allowed to send subsequent messages or should expect another message. Therefore, the conversation defines a *conversation state diagram* constructed by nodes, denoting conversation states, and edges, representing messages sent between endpoints. The role of messages is to transfer business-logic data as well as to carry conversation state transitions between endpoints, hence synchronizing conversation state in endpoints. A message can be sent only if the sending endpoint has control over the conversation and the conversation state diagram allows for sending such a message in a given state.
3. The conversation state diagram has to be deterministic; that is, there should be only one endpoint which has control over the conversation in a given state, no matter which transition sequence has led to this state.
4. Endpoints can pass messages to each other in an asynchronous and non-blocking way: the sending operation returns once the message is successfully committed to the medium. Messages delivered by the medium are placed in the buffer queue and processed sequentially by the recipient endpoint in the same dedicated thread. Therefore, the conversation is driven by a pair of *conversation threads*, one per each endpoint.
5. Conversation thread being a mediator in message delivery is the entity that takes care of timing and/or reliability aspects, i.e., it can decide whether to suspend the endpoint processing by holding up sending operation return or whether to postpone or give up sending messages according to some timing policy (e.g. jitter compensation policy).

6. The conversation is initiated by starting both endpoints' conversation threads. Both endpoints are initially in the *init* state. The client's conversation thread starts the exchange by sending the first message to the server.
7. The conversation stops as soon as it reaches the *final* state and all messages are processed by conversation threads.
8. Messages can also set up *sub-conversations* whose life-cycle is contained in the scope of a single state of the *parent conversation*. Sub-conversations can be created by an endpoint only if it currently has control over the conversation. A *sub-conversation handle* can be passed to a remote endpoint along with a message. The recipient endpoint can use this handle in order to start a sub-conversation. The sub-conversations allow for modularization and decomposition of the complex conversation schemes into fine-grained, easily maintainable parts and consequently enables tree-like scaling to larger and more complex conversation patterns.

4 Conversation Protocol Meta-model

An important feature of CompTalks is that it proposes a way of modelling conversation protocols. Once modelled, a protocol can be subjected to analysis, simulation and validation.

The conversation Protocol Meta-Model defines a *Conversation* that comprises *Client* and *Server Endpoints*. Each endpoint specifies a set of *Messages* it accepts. Each message, in turn, is defined by its *Signature*, *Required State*, *Implied State* and *Control Passing Flag*. This can be formalized using a UML class diagram, as presented in Fig. 1.

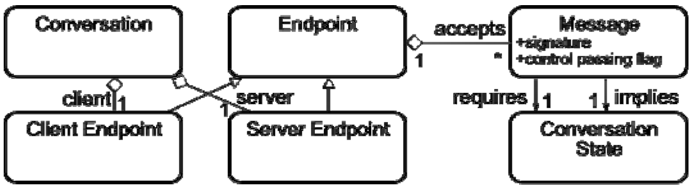


Fig. 1. Conversation Protocol Meta-Model expressed as a UML class diagram

For example let us examine a simple conversation depicted as a UML object diagram in Fig. 2 (a). The conversation starts when the client sends the *foo* message to the server. The *foo* message conducts *text* data of type *String* and causes transition from *init* to *intermediate* conversation state. It also hands over the control over the conversation to the server. After successfully committing the *foo* message to the medium the client enters the *intermediate* state and loses control over the conversation. After receiving the message, the server takes control over the conversation and enters the *intermediate* state where it processes the received data. Having control and being in the *intermediate* state, the server is able to send the *bar* message, which carries *text* data of type *String*, causing transition from the *intermediate* to the *final* state and handing control over to the client. Once the server successfully commits the *bar* message to the medium, it enters the *final* state and hence the conversation thread on the server side ends. Once the client receives the message, it similarly enters the *final* state and processes the received data. After the data is processed, the conversation thread on the client side ends. A sequence diagram describing such a course of this sample conversation is shown in Fig. 2 (b).

Owing to its intrinsic scalability, CompTalks allows for creating more robust and complex conversation protocols. It supports conversation state diagrams of unbounded finite sizes and provides mechanisms for nesting an unbounded finite number of sub-conversation within the scope of a parent conversation. The conversations are then organized in a tree-like structure, where conversation lifecycle of child conversation is contained in the scope of one of the states of the parent conversation, while sibling conversations' lifecycles are independent and proceed in parallel.

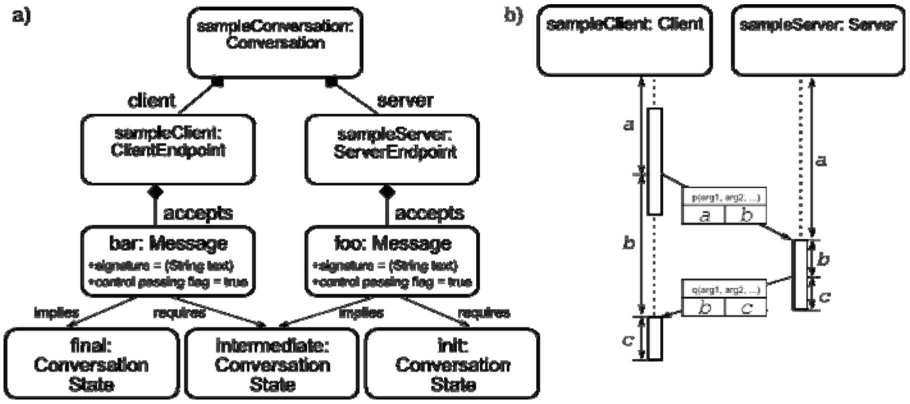

a)
sampleConversation: Conversation
client
server
sampleClient: ClientEndpoint
sampleServer: ServerEndpoint
accepts
accepts
bar: Message
+signature = (String text)
+control passing flag = true
foo: Message
+signature = (String text)
+control passing flag = true
implies
requires
implies
requires
final: Conversation State
intermediate: Conversation State
init: Conversation State
b)
sampleClient: Client
sampleServer: Server
a
a
b
b
c
c
p(arg1, arg2, ...)
a
b
q(arg1, arg2, ...)
b
c

Fig. 2. Sample conversation model that conforms to the Conversation Protocol Meta-Model expressed as a UML object diagram (a) along with a sequence diagram of the course of this conversation (b)

4.1 Methods for Analysis and Validation of Conversation Protocol Models

The Conversation Protocol, modelled in the aforementioned way, can be subject to analysis, simulation and, eventually, validation in terms of conformance to CompTalks conversation rules. This section shows how to transform the conversation protocol into a powerful and thoroughly explored Petri Net [15] model which enables studying static and dynamic properties of asynchronous and concurrent systems.

Transformation from the Conversation Protocol Model to the Petri Net model involves tree steps:

1. Transforming each message specification into a subnet of places and transitions according to the following rules:
   a. Message *m* accepted by the server, with conversation transition from state *p* to *q* and with control passing, results in the subnet depicted in Fig. 3 (a)
   b. Message *m* accepted by the server, with conversation transition from state *p* to *q* and without control passing, results in the subnet depicted in Fig. 3 (b)
   c. Message *m* accepted by the client, with conversation transition from state *p* to *q* and with control passing, results in the subnet depicted in Fig. 3 (c)

   d. Message *m* accepted by the client, with conversation transition from state *p* to *q* and without control passing, results in the subnet depicted in Fig. 3 (d)

2. Merging all subnets obtained in step 1.
3. Setting initial marking by placing tokens in *client state initial*, *server state initial* and *client control* places.

According to the transformation procedure, for the conversation composed of *m* messages and *s* states the resulting Petri Net is consisting of *2*s+m+2* places and *2*m* transitions, therefore the size and complexity of a net is merely linearly correlated with a number of states and messages of a conversation. The sample protocol described in the previous section can be represented as a Petri Net, as shown in Fig. 3 (e).

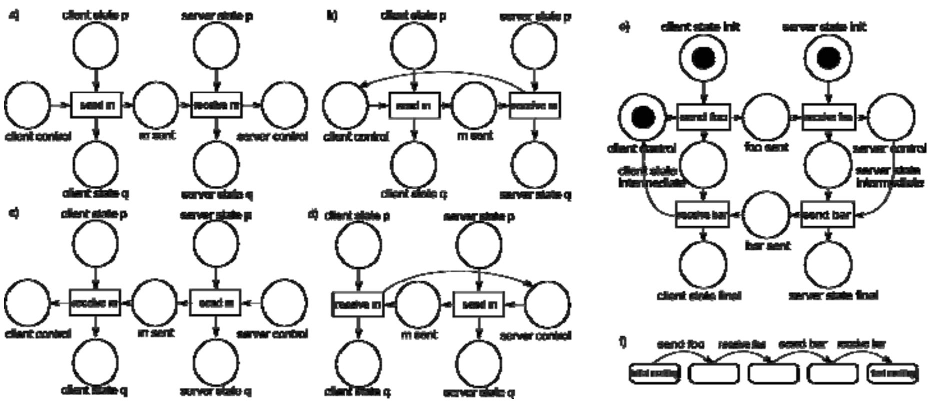


Fig. 3. Conversation Protocol Model to Petri Nets model transformation rules (a-d); sample protocol model transformed to the Petri Nets model (e) and its marking graph (f)

For the obtained Petri Net, a marking graph can be computed and subsequently analyzed. The conversation is considered valid if and only if all the following constraints are met by the marking graph:

1. Each *send X* transition is followed only by *receive X* transition.
2. Each *receive* transition is followed only by zero or more *send* transitions.
3. The only dead marking is the *final marking* with tokens placed in *client state final*, *server state final* and *client control* places.
4. The *final marking* is reachable from each marking that is reachable from the *initial marking*.

As the marking graph for the sample protocol shown in Fig. 3 (f) meets all above-listed requirements the conversation is considered valid.

Such analysis and validation on the model level is useful in ensuring correctness, as it takes place on an early stage of design and at a high level of abstraction. The presented method is abstract and can be implemented using notations specific to a particular framework, e.g. the reference implementation of the CompTalks Framework presented below.

5 CompTalks Framework

The CompTalks Framework is a Java-based reference implementation of CompTalks. The conversation protocols are specified by a pair of Java interfaces which extend *ServerEndpoint* and a *ClientEndpoint* interfaces respectively and specify methods responsible for message interception. Such methods are to be annotated with a *Transition* annotation, which, in turn, specifies the required state of a message (*from* field), its implied state (*to* field) and the control passing flag (*control* field). As an illustration, the CompTalks specification of the sample conversation protocol from Fig. 2 is shown in Fig. 4.

```
public interface SampleServer extends ServerEndpoint<SampleClient> {
   @Transition(from = Transition.INITIAL_STATE, to = "intermediate",
   control = true)
   public void foo(String text);
}
public interface SampleClient extends ClientEndpoint<SampleServer> {
   @Transition(from = "intermediate", to = Transition.FINAL_STATE,
    control = true)
   public void bar(String text);
}
```

Fig. 4. *SampleServer* and *SampleClient* interfaces (irrelevant code fragments omitted)

Applied annotation approach allows for keeping the specification of the protocol in a single source code entity, namely Java interface, and in a single binary entity of Java class file what keeps source code self-documenting on the one hand, and binary file self-contained and easily distributable on the other.

A pair of interfaces, being a specification of a conversation protocol, can be subject to conversation protocol model analysis and validation. The CompTalks Framework offers an analyzer, that, given a pair of compiled interfaces and using the Java Reflection API, determines whether the model is valid according to the Petri Net method presented in the previous section.

Endpoint interfaces' definitions should be provided to the client and the server sides and realized by concrete endpoint implementations, such as the ones presented in Fig. 5. Implementation can be examined in terms of conforming to a given model, e.g., whether implementation ensures that messages are sent only in allowed conversation states etc. This kind of validation may be performed by applying bytecode analysis techniques to the code of implementation classes. Such two-level validation (model validation followed by implementation validation) allows for rapid and early evaluation of software correctness. Moreover, as validation takes place at compile time, it greatly tightens the development-test-feedback loop and eliminates performance-consuming correctness checking at runtime.

The CompTalks Framework is currently available for use and offers support for two types for message passing media: TCP-based and TCP/TLS-based media, where the latter uses TLS [16] for the purpose of authentication and maintaining the confidentiality of exchanged data. Both basic and complex data types in message signatures including Java basic types and Plain Old Java Objects (POJOs) are supported and serialized to XML through standard Java API for XML Binding (JAXB) [17]. Conversation Protocol Model Analyzer is able to examine protocols

```
public class SampleServerImpl implements SampleServer {
   private SampleClient sampleClient;
   public void setRemoteEndpoint(SampleClient remoteEndpoint) {
      this.sampleClient = remoteEndpoint;
   }
   public void foo(String text) {
    // entered 'intermediate' state
      this.sampleClient.bar("some business logic data");
    // entered 'final' state
   }
}
public class SampleClientImpl implements SampleClient {
   private SampleServer sampleServer;
   public void setRemoteEndpoint(SampleServer remoteEndpoint) {
      this.sampleServer = remoteEndpoint;
   }
   public void start() {
    // entered 'init' state
      this.sampleServer.foo("some business logic data");
    // entered 'intermediate' state
   }
   public void bar(String bar) {
    // entered 'final' state
   }
}
```

Fig. 5. Sample realizations of *SampleServer* and *SampleClient* interfaces (irrelevant code fragments omitted)

modelled as a pair of endpoint interfaces in terms of validity, based on the Petri Net formalism. The current distribution of CompTalks Framework includes command-line tools for running server and client endpoints and a set of sample demo protocols.

6 Validation and Tests

The CompTalks framework has been developed in the course of the ViroLab [2] Virtual Laboratory [3] project. CompTalks-powered GSEngine [1] forms the core of the Virtual Laboratory and enacts scientific workflows called *experiments*. These experiments are expressed as Ruby [18] scripts and are capable of accessing resources (such as data sets and services), interact with human actors and provide rich end-user experience. Enactment of scripts by a remote GSEngine service involves complex client-server interaction, including: streaming standard input and output, serving input forms to end users, uploading and downloading data and script files, tracing the status of experiments, sending results, notifications etc. All these features have been successfully implemented with CompTalks. Satisfactory stability and performance of the solution has allowed for employment of CompTalks in the production setup of the Virtual Laboratory.

Estimation of the performance of the CompTalks framework was carried out by measuring the time and memory required by the client in order to carry out a number of simultaneous benchmark conversations. Since the sample conversation discussed in this paper represents the simplest bidirectional message exchange, and its reference implementation consumes as little time and memory as possible, it is a useful benchmark for estimation of the overhead introduced by the framework itself.

The test results, presented in Fig. 6, were obtained on a testbed constituted by the 64-bit Sun Blade server, with dual-core Intel Xeon 5150 2.66GHz CPU running Ubuntu OS version 4.0.3 (Linux version 2.6.15-28-amd64-server) and HotSpot Java Virtual Machine version 1.6.0 update 10. Both client and server were running on the same machine and were communicating through the *localhost* network interface (RTT latency measured using ping equal to 0,006±0,001ms), via a plain TCP medium. The obtained results show that time and memory consumption is linearly dependent on the number of simultaneous conversations. Time consumption estimation is 235ms (bias) plus 1,6ms per each conversation. Memory consumption is estimated as 14,8MB (bias) plus 0,09MB per each conversation.

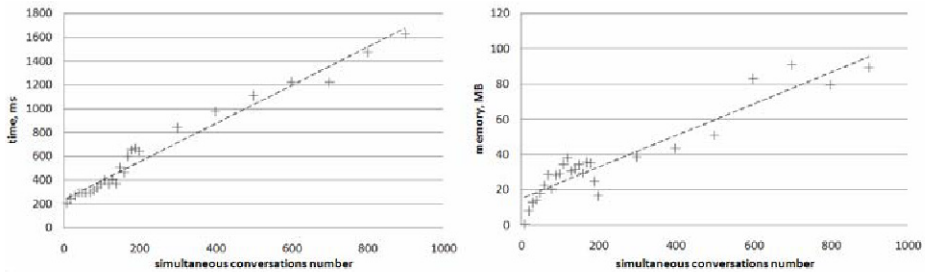


Fig. 6. Performance test results: time and memory required by the client to simultaneously carry out a number of benchmark conversations in the testbed environment

7 Conclusions and Future Prospects

The research described in this paper has shown that vital design and implementation issues related to a certain class of services, thus far unaddressed, can be remedied by using the proposed conversation protocols model. The example of GSEngine and ViroLab shows that this concept is applicable, and establishes the framework as a convenient and productive software development facility as well as effective middleware technology.

As we find this concept worth further research, the development of CompTalks is still ongoing and the following challenges are expected to be addressed next:

- Support for distribution, clustering and load balancing between servers by employing a master-worker architecture in order to make the solution scalable in terms of throughput (currently at a prototype stage).
- Enabling the Conversation Protocol Endpoint Implementation Analyzer to examine whether the endpoint implementation meets a given conversation protocol model. This will be based on ASM [19] Java Virtual Machine (JVM) [20] bytecode analysis and manipulation library (currently at a proof-of-concept stage).

The other further areas of research can be targeted at supporting multi-actors, multicast conversation and development of rich, interactive GUI design patterns that would integrate seamlessly with the proposed Conversation Meta-Model.

Acknowledgments. This work was partly funded by the European Commission, Project ViroLab IST-027446 and the related Polish grant SPUB-M, the AGH grant

11.11.120.777, and ACC CYFRONET-AGH grant 500-08. The authors wish to thank Tomasz Gubała and Piotr Nowakowski for their valuable advice.

References

1. Ciepiela, E., Kocot, J., Gubala, T., Malawski, M., Kasztelnik, M., Bubak, M.: Virtual Laboratory Engine – GridSpace Engine. In: Cracow Grid Workshop 2007 Workshop Proceedings, ACC CYFRONET AGH, pp. 53–58 (2008)
2. ViroLab - virtual laboratory for decision support in viral deseases treatment, `http://virolab.org/`
3. Bubak, M., et al.: Virtual Laboratory for Development and Execution of Biomadical Collaborative Applications. In: Puuronen, S., Pechenizkiy, M., Tsymbal, A., Lee, D.-J. (eds.) Proc. 21st IEEE International Symposium on Computer-Based Medical Systems, Jyvaskyla, Finland, June 17-19, pp. 373–378 (2008) doi 10.1109/CBMS.2008.47
4. World Wide Web Consortium (W3C) Web Services activity, `http://www.w3.org/2002/ws/`
5. Web Services Description Language (WSDL) Specification 1.1, W3C Note (March 15, 2001), `http://www.w3.org/TR/wsdl`
6. Web Services Conversation Language (WSCL) 1.0, W3C Note (March 14, 2002), `http://www.w3.org/TR/wscl10/`
7. Simple Object Access Protocol (SOAP) 1.1, W3C Note (May 8, 2000), `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`
8. Web Services Resource (WS-Resource) 1.2, OASIS Standard (April 1, 2006), `http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf`
9. Web Services Transfer (WS-Transfer),W3C Member Submission (September 27, 2006), `http://www.w3.org/Submission/WS-Transfer/`
10. Fielding, R.T., Taylor, R.N.: Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology (TOIT) 2(2), 115–150 (2002) doi:10.1145/514183.514185
11. Foster, I., Parastatidis, S., Watson, P., McKeown, M.: How do I model state?: Let me count the ways. Commun. ACM 51(9), 34–41 (2008)
12. Extensible Messaging and Presence Protocol XMPP home page, `http://xmpp.org/`
13. Extensible Messaging and Presence Protocol (XMPP): Core, IETF RFC 3920 (October 2004), `http://www.ietf.org/rfc/rfc3920.txt`
14. The Blocks Extensible Exchange Protocol Core, IETF RFC 3080 (March 2001), `http://www.ietf.org/rfc/rfc3080.txt`
15. Peterson, J.L.: Petri Nets. ACM Computing Surveys 9(3), 223–252 (1977) doi:10.1145/356698.356702
16. Transport Layer Security (TLS), IETF RFC 5246 (August 2008), `http://www.ietf.org/rfc/rfc5246.txt`
17. JSR-222 Java Architecture for XML Binding (JAXB), `http://jcp.org/aboutJava/communityprocess/mrel/jsr222/`
18. Ruby Programming Language, `http://www.ruby-lang.org/en/`
19. ASM - all purpose Java bytecode manipulation and analysis framework, `http://asm.ow2.org/`
20. Lindholm, T., Yellin, F.: The Java Virtual Machine Specification, 2nd edn.
21. Rosmanith, H., Volkert, J.: glogin - Interactive Connectivity for the Grid. In: Juhasz, Z., Kacsuk, P., Kranzlmüller, D. (eds.) Distributed and Parallel Systems - Cluster and Grid Computing, Proc. of DAPSYS 2004, 5th Austrian-Hungarian Workshop on Distributed and Parallel Systems, September 2004, pp. 3–11. Kluwer Academic Publishers, Budapest (2004)

CAMEO: Continuous Analytics for Massively Multiplayer Online Games on Cloud Resources*

Alexandru Iosup

Electrical Eng., Mathematics and Computer Science Department
Delft University of Technology, Delft, The Netherlands
A.Iosup@tudelft.nl

Abstract. Massively Multiplayer Online Games (MMOGs) have grown to entertain tens of millions of players daily. Currently, the game operators and third-parties using gameplay information rely on pre-provisioned resources to analyze the current status of the player community and the evolution of this status over time. Instead, with the appearance of cloud computing it has become attractive to use on-demand resources to run automated MMOG data analytics tools. Thus, in this work we introduce CAMEO, an architecture for Continuous Analytics for Massively multiplayEr Online games on cloud resources. Our architecture provides various mechanisms for MMOG data collection and continuous analytics of a pre-determined accuracy in real settings. We assess the capabilities of our approach by taking and analyzing complete or partial snapshots from Runescape, one of the most popular MMOGs with a community of over 3,000,000 active players. Notably, we show evidence that CAMEO already supports simple continuous MMOG analytics, and give a first estimation of the costs of the analytic process.

1 Introduction

Massively Multiplayer Online Games (MMOGs) gather tens of millions of players into a fractioned online community. To serve the interests of these players, the game operators and the third-party entities such as community and fan-owned web sites need to collect, analyze, and then synthesize the status of the community components. While the final synthesis may differ from entity to entity, the data collection and analysis (collectively, the *game analytics*) can benefit from recent advances in the availability of on-demand resources through cloud computing services such as Amazon's Elastic Compute Cloud (EC2). In this work we present CAMEO, an architecture for continuous analytics of data taken from massively multiplayer online games on cloud resources.

Online data crawling has often been employed in the past to determine the stationary and dynamic characteristics of Internet-based communities. However, the focus of the research community has been either in making the crawling process more parallel [1, 2, 3], or analyzing the acquired data using more scalable parallel or distributed algorithms [4, 5]; both these approaches assume that

* We thank the Delft ICT Talent Grant for the financial support.

enough resources are available for the task. In contrast, in this work we focus on a domain-specific application, MMOGs, and focus on a different problem which stems from a restricted resource availability (which in turn is the direct result of minimizing costs): continuous analytics of a pre-determined accuracy in real settings. Our contribution is threefold:

1. We present a first formulation of the problem of continuous analytics for MMOGs (Section 2);
2. We introduce CAMEO, an architecture for continuous analytics of data taken from massively multiplayer online games that uses cloud computing environments to dynamically obtain resources(Section 3);
3. We show that CAMEO can be used to acquire and track data from Runescape, a popular MMOG, and give a first cost estimation for this process (Section 4).

2 Continuous Analytics for MMOGs

In this section we present the problem of continuous analytics for MMOGs.

2.1 Definition

MMOGs generate data that need to be analyzed at various levels of detail and for various purposes, from high-level analysis of the number of players in a community for in-game reward allocation to the detailed analysis of the user mouse clicking behavior for audit and cheat detection. Usually, a replica of the data to be analyzed needs to be created, which raises the problem of maintaining consistency between the original and the replica(s). Similar to other cases of information replicas in distributed systems, creating exact copies of the data for analysis purposes may not be only expensive, but also unnecessary [6]. Instead of ensuring that the replicas are strongly consistent, our goal is to maintain information replicas whose difference is bounded and the bound is under the control of the analyst. This goal stems from traditional work on continuous consistency of information replicas with deviation in the staleness of information [6] and quasi-copying [7].

We can now define *continuous analytics for MMOGs* as the process through which relevant MMOG data are analyzed in such a way that prevents the loss of important events affecting the data. The relevance of the data is application-specific, as it depends on the target of the analysis. Similarly, the important events allow for the information replicas to be loosely consistent with the original, within application-specific bounds.

2.2 Challenges

Every data analysis process includes data collection, storage, processing, and presentation, each of which raises generic challenges in supporting continuous

MMOG analytics. We focus here only on the MMOG-specific challenges, challenges due to data characteristics, and challenges due to data ownership, which we describe in turn.

MMOGs pose unique data scale and rate challenges. MMOGs generate and manage massive amounts of information; for example, the database logging user actions for Everquest 2, a popular MMOG, stores over 20 new terrabytes (TB) of data per year. Other projects such as CERN's Large Hedron Collider or the Sloan Digital Sky Survey produce data orders of magnitude larger than MMOGs, but these projects are using large and pre-provisioned (expensive) computational and data infrastructure that game companies cannot afford. Furthermore, the data production rate for these other projects is stable over time spans of days or even weeks, whereas for MMOGs the daily user activity has peaks and may even change hourly [8].

MMOGs pose unique data ownership challenges. MMOGs often involve multiple companies in their design-development-distribution-use process; each of these companies may have different commercial interest and thus compete for generating and managing game-related data. Moreover, there may be many types of data users, from audit companies who should access all data to fan communities that may only be allowed to access information open to everyone. Thus, MMOGs raise data access challenges. Other commercial applications, notably financial and government public relations services, face similar problems. However, in contrast to MMOGs these services produce data for entities that can afford expensive data collection and processing infrastructure, such as brokering agencies or news corporations.

2.3 Applications

There are many applications for continuous MMOG analytics, both for the gaming industry and for other domains. We describe the most important such applications in the following.

Within the gaming industry, the main applications are to audit the process of each company involved with the MMOG, to understand the play patterns of users and support future investment decisions, to detect cheating and prevent game exploits, to provide user communities with data for ranking players, to broadcast gaming events, and to produce data for advertisement companies and thus increase the revenue stream for the MMOG owners.

In other areas, by domain the applications may include studying emergent behavior in complex systems (systems theory), understanding the emergence and evolution of the contemporary society [9] (social sciences) and economy [10] (economics), uncovering the use of MMOGs as cures and coping mechanisms [11] (psychology), investigating disease spread models [12] (biology), etc.

3 The CAMEO Architecture

In this section we present the CAMEO architecture for continuous MMOG analytics. The CAMEO architecture is built around the idea of enabling continuous

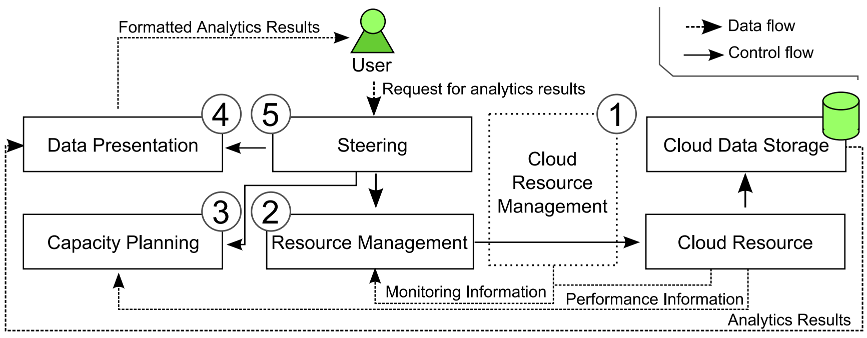


Fig. 1. The CAMEO architecture

MMOG analytics while using resources only when needed. To achieve this goal, it acquires and releases computational and storage resources dynamically from cloud computing environments such as Amazon's EC2+S3.

3.1 Overview

The five main components of the CAMEO architecture are depicted in Figure 1. The *Cloud Resource Management* component (component 1 in Figure 1) provides access to the computational and storage resources of the cloud computing environment, and is maintained by the cloud owner. The *Resource Management* component (#2) acquires and releases resources from the cloud and runs the analytics applications. It also uses the monitoring information provided by the cloud resource management and the resources as input for further management actions, such as transparent fault tolerance through application instance replication. The *Capacity Planning* component (#3) is responsible for deciding how many resources must be acquired for the analytics process. The decisions are based on the system's capability to produce results, analyzed during the course of the analytics process, and on the accuracy and cost goals of the process. The *Data Presentation* component (#4) formats and presents the results of the analytics process to the user. The *Steering* component (#5) is responsible for coordinating the analytics process. Towards this end, it takes high-level decisions, expressed through the configuration of each other's component process.

Except for the use of cloud computing resources, our architecture uses a traditional approach. However, the components have unique features specific to the targeted application. We describe in the remainder of this section three distinctive features of CAMEO.

3.2 Resource Management Mechanisms

The triggering of the analytics process depends on the nature of the application and on the system status. On the one hand, the nature of the application may allow the system analyst to design a stable analysis process such as a daily investigation of the whole community of players. On the other hand, special analysis may be required when the system is under unexpectedly heavy load, or when many players are located in the same area of the virtual world. To address this situation, we design the Resource Management component to provide two mechanisms for using cloud resources: one static and one dynamic. The *steady analytics*[1] mechanism allows running a periodic analytics operation on cloud resources. The *dynamic analytics* mechanism allows running a burst of analytics operations on cloud resources. Optimizing the allocation of resources for static analytics or for mixed static-dynamic analytics is a target for this component, but beyond the scope of this work. Similarly, the case when the cost of data transfers is significant, that is, similar or higher to the cost of the computational resources, is left for future work.

3.3 Steering through Snapshots of Different Size

The analytics process includes collecting the necessary information from the data source. The collection results in a *snapshot*, that is, a read-only dataset which has been extracted from the original data. We further call *complete snapshot* a snapshot that includes data for all the players managed by the MMOG, and contrast it to a *partial snapshot.* Taking snapshots complies with the continuous analytics definition introduced in Section 2.1.

Depending on the goal of the analysis, it may be possible to obtain meaningful results through continuous analytics based on partial snapshots; for example, when the goal is to obtain statistical information about the player community it may suffice to continuously analyze a randomly chosen group of players of sufficient size. We design the Steering component to be able to perform a two-step analytics process in which first complete snapshots are taken from the system with low frequency, and partial snapshots are acquired often.

3.4 Controlling the Process

The taking of a snapshot has a certain duration, which depends on the performance of the cloud resources and also on the limitations set by the owners of the original data; to prevent denial-of-service attacks and to improve scalability with the number of requests, it is common for the data owners to limit the network bandwidth available for an individual resource (IP address).

Assume that a single machine can acquire a new snapshot every T time units (seconds). Then, we can achieve linear scaling (to a certain degree) in the number of acquired snapshots by installing new machines; K machines can acquire K snapshots every T time units. We can then control either how many snapshots we acquire every T time units, or the minimal performance that has to be delivered by each machine to acquire exactly one snapshot every T time units.

[1] We do not use the term "static" to underline that this is a continuous process.

Table 1. The resource characteristics for the instance types offered by Amazon EC2

Resource Type	Cores (ECUs)	RAM [GB]	Architecture [bit]	I/O Performance	Disk [GB]	Cost [$/h]
m1.small	1 (1)	1.7	32	Med	160	0.1
m1.large	2 (4)	7.5	64	High	850	0.4
m1.xlarge	4 (8)	15.0	64	High	1,690	0.8
c1.medium	2 (5)	1.7	32	Med	350	0.2
c1.xlarge	8 (20)	7.0	64	High	1,690	0.8

4 Experimental Results

In this section we show evidence that our approach (and CAMEO implementation) can be used for continuous MMOG analytics. (Analyzing the results of a continuous MMOG analytics process falls outside the scope of this work.)

4.1 Experimental Setup

The Analyzed MMOG. Using CAMEO, we have taken and analyzed several complete snapshots of the state of Runescape over a period of one and a half years. We have also also taken partial snapshots of the state of Runescape in quick succession, which enables us to study in the future the dynamics present in the Runescape community. We have written application-specific web crawlers for the data collection process.

The Platform. We have used Amazon EC2 resources to acquire and process Runescape data. The EC2 user can use any of the five resource (*instance*) types currently available on offer, the characteristics of which are summarized in Table 1. An ECU is the equivalent CPU power of a 1.0-1.2 GHz 2007 Opteron or Xeon processor. The theoretical peak performance can be computed for different instances from the ECU definition: a 1.1 GHz 2007 Opteron can perform 4 flops per cycle at full pipeline, which means at peak performance one ECU equals 4.4 gigaflops per second (GFLOPS). Throughout the experiments conducted for this work we have used the `m1.small` instances; extending the Capacity Planning module with the ability to use multiple instance types is left as future work.

4.2 Analytics Results

Using CAMEO, we analyzed the skill level of millions of RuneScape players, which shows evidence that CAMEO can be used for measurements several orders of magnitude larger than the previous state-of-the-art [13]. CAMEO collected in August 2008 official skill level data for 2,899,407 players[2], of which 1,817,211 (over 60%) had a skill level above 100; the maximum skill level is 2280. The values for players with skill level below 100 include application-specific noise (mostly starting players) and are therefore polluted. Thus, we present here only data for

[2] The current population of Runescape has increased to over 3,000,000 active players.

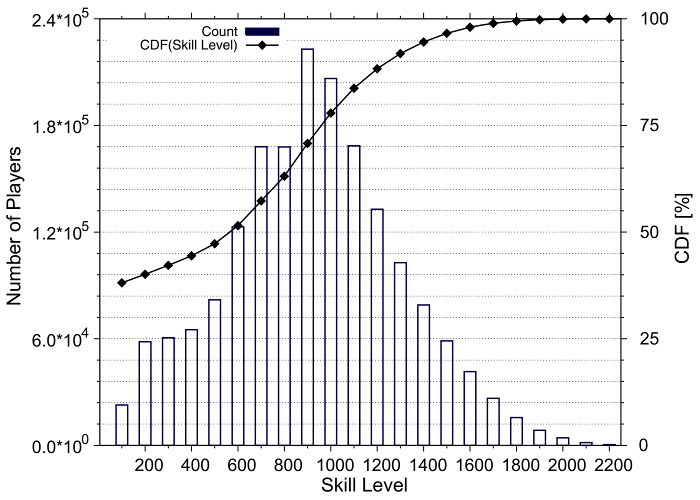


Fig. 2. Pareto graph, that is, combined PDF (left vertical axis) and CDF (right vertical axis) depiction of the skill level of the RuneScape player population. Each bar represents a range of 100 levels. CDF stands for cumulative distribution function; $CDF(x)$ is the total number of players with skill level up to and including x. Note that the left vertical axis is not linear. See text for why the CDF of the skill level does not start at 0%.

all players with skill above 100, and for a single measurement. Figure 2 depicts the overall skill level of RuneScape players, with bins of 100 levels. The number of players per bin is well characterized by a skewed normal-like distribution; the majority of the players are of average skill or below, the most populated skill level bins are those corresponding to the middle skill level, and the number of high-level players is significant. We have explored the implications of this skill level distribution in our previous work on automatic content generation [14].

4.3 Resource Management

To demonstrate the capability of CAMEO to perform both dynamic and steady analytics, and to monitor the process, we show in Figure 3 the evolution of the cumulative number of consumed CPU hours over time. The dynamic analytics are based on uneven bursts of activity, of which the burst during March 10 is the most prominent. The steady analytics part of the experiments reveals an even use of resources over time, with the steps indicating a new work cycle.

One of the contributions of this work is getting a first estimation on the cost of continuous MMOG analytics. Figure 4 shows the total cost incurred by the continuous analytics process over the course of one month. For this simple analysis process, which acquired partial snapshots and only browsed the data in memory during the processing phase, the cost is below \$500 per month. It is not our intention to argue that the cost of continuous analytics for an MMOG

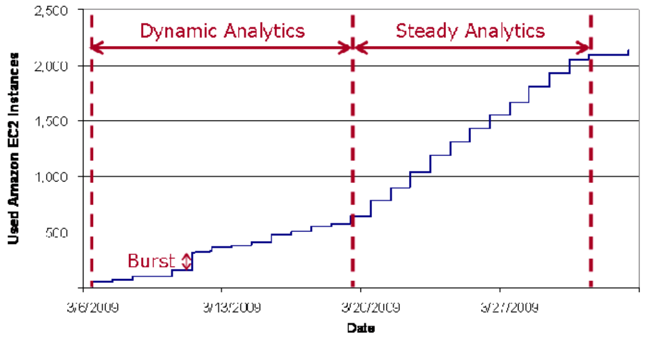
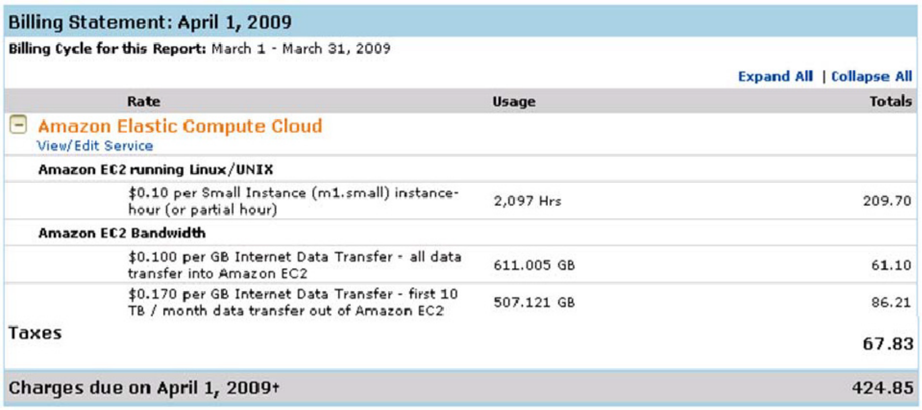


Fig. 3. Resource consumption in the two analytics modes: dynamic and static



Billing Statement: April 1, 2009

Billing Cycle for this Report: March 1 - March 31, 2009

Expand All | Collapse All

Rate	Usage	Totals
Amazon Elastic Compute Cloud View/Edit Service		
Amazon EC2 running Linux/UNIX		
$0.10 per Small Instance (m1.small) instance-hour (or partial hour)	2,097 Hrs	209.70
Amazon EC2 Bandwidth		
$0.100 per GB Internet Data Transfer - all data transfer into Amazon EC2	611.005 GB	61.10
$0.170 per GB Internet Data Transfer - first 10 TB / month data transfer out of Amazon EC2	507.121 GB	86.21
Taxes		**67.83**
Charges due on April 1, 2009+		**424.85**

Fig. 4. Putting a cost on continuous analytics for MMOGs

can be this low; much more complex analytics taking many more computational hours are performed for any of the applications presented in Section 2.3.

4.4 Platform Capabilities

An important assumption in our work is that resources can be acquired on-time, that is, that whenever resources are requested by the Resource Management component of CAMEO the cloud will provide them within a reasonable time. We now show that this is indeed the case.

We have made initial install time measurements in August 2007, and found that the average install time was steady around 50s [15]. To understand the long-term evolution of the install time in EC2, we have obtained the measurement results published online by the independent CloudStatus team [16]. We have written web crawlers and parsing tools and taken samples every two minutes from August until October 2008 (set #1, two months), and from December 2008 until May 2009 (set #2; only the first four months are depicted in Figure 5).

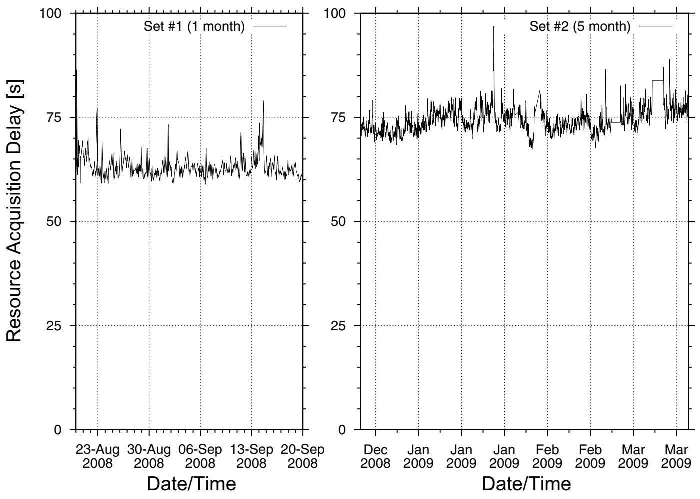


Fig. 5. Evolution of the VM Install time in EC2 (hourly average) over six months (two data sets collected from `CloudStatus.com`)

Figure 5 shows that the install time fluctuates by around 5 seconds within short time intervals (days), but that the average install time has increased from 50s in August 2007, to 64s in August 2008, and to 78s in April 2009. This indicates a doubling of the rate of the increase in install time every half year. If this trend continues, conservatively the install time will reach 80s in 2009 (confirmed), and around two minutes by June 2010. We leave as future work a detailed study of the time patterns that may occur in the install time, e.g., effects of the hour-of-the-day and of the day-of-the-week. We conclude that the resource acquisition time is steady within a short period of time (hours, days) and has a slow yearly increase. An investigation of the storage capabilities of the Amazon cloud [17] allows us to reach the conclusion that the use of cloud resources for continuous MMOG analytics is possible even for bursts of user activity.

5 Related Work

We have already discussed the main differences between our work and generic web crawling approaches [1, 2, 3] and parallel or distributed analytics [4, 5]. In contrast with this body of previous research, ours focuses on a more restricted application–albeit with millions of users– but focuses on using (and paying for) the resources used in the analytics process only when they are needed.

Closest to our work, Provost and Kolluri [4] examine many basic techniques for scaling up inductive algorithms. While data analytics (as a superset of inductive algorithms) has evolved considerably in the decade passed since this survey, the problem of continuous MMOG analytics raises new challenges, and our approach is based on using on-demand resources (cloud computing) instead of a fixed computational platform.

6 Conclusion and Future Work

The growing world of Massively Multiplayer Online Games (MMOGs) raises important derivative online applications and interesting new challenges to the distributed computing community, including the problem of massive game data analytics. Motivated by a subset of this problem, in this work we have introduced CAMEO, an architecture for continuous analytics of data taken from massively multiplayer online games on cloud resources. Using a reference implementation of CAMEO and resources leased from the Amazon EC2 cloud, we have taken complete and partial snapshots of the Runescape multi-million player community for a period of over eighteen months. Our results give evidence that cloud computing resources can be used for continuous data acquisition and analysis. Furthermore, we have devised within CAMEO mechanisms for controlling the analytics process, including taking partial snapshots of a given size, whose analysis leads to a pre-determined accuracy of the results. Last, we have provided a first cost estimation for the continuous analytics process.

For the future, we plan to investigate in more detail the trade-off between the amount of data acquired and the quality of the analysis results for MMOGs. We will also investigate the use of more heterogeneous resource types coming from one or more clouds, and the restricted use of cloud resources when local resources are available.

References

1. Arasu, A., Cho, J., Garcia-Molina, H., Paepcke, A., Raghavan, S.: Searching the web. ACM Trans. Internet Technol. 1(1), 2–43 (2001)
2. Cho, J., Garcia-Molina, H.: Parallel crawlers. In: WWW, pp. 124–135 (2002)
3. Lee, H.-T., Leonard, D., Wang, X., Loguinov, D.: Irlbot: scaling to 6 billion pages and beyond. In: WWW, pp. 427–436 (2008)
4. Provost, F.J., Kolluri, V.: A survey of methods for scaling up inductive algorithms. Data Min. Knowl. Discov. 3(2), 131–169 (1999)
5. Bayardo, R.J., Ma, Y., Srikant, R.: Scaling up all pairs similarity search. In: WWW, pp. 131–140 (2007)
6. Yu, H., Vahdat, A.: Efficient numerical error bounding for replicated network services. In: VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases, pp. 123–133. Morgan Kaufmann Publishers Inc., San Francisco (2000)
7. Alonso, R., Barbará, D., Garcia-Molina, H.: Data caching issues in an information retrieval system. ACM Trans. Database Syst. 15(3), 359–384 (1990)
8. Nae, V., Iosup, A., Podlipnig, S., Prodan, R., Epema, D.H.J., Fahringer, T.: Efficient management of data center resources for massively multiplayer online games. In: ACM/IEEE SuperComputing, IEEE/ACM (2008)

9. Steinkuehler, C., Williams, D.: Where everybody knows your (screen) name: Online games as "third places". In: DIGRA Conf. (2005)
10. Castronova, E.: On virtual economies. Game Studies 3(2) (2003)
11. Williams, D., Yee, N., Caplan, S.: Who plays, how much, and why? debunking the stereotypical gamer profile. Journal of Computer-Mediated Communication 13(4), 993–1018 (2008)
12. BBC NEws, Virtual game is a 'disease model', News Item (September 2009), `http://news.bbc.co.uk/2/hi/6951918.stm`
13. Yee, N.: The demographics, motivations, and derived experiences of users of massively multi-user online graphical environments. Presence 15(3), 309–329 (2006)
14. Iosup, A.: POGGI: Puzzle-based Online Games on Grid Infrastructures. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009 Parallel Processing. LNCS, vol. 5704, pp. 390–403. Springer, Heidelberg (2009)
15. Iosup, A., Tannenbaum, T., Farrellee, M., Epema, D.H.J., Livny, M.: Inter-operating grids through delegated matchmaking. Scientific Programming 16(2-3), 233–253 (2008)
16. The Cloud Status Team, JSON report crawl (January 2009), `http://www.cloudstatus.com/`
17. Palankar, M.R., Iamnitchi, A., Ripeanu, M., Garfinkel, S.: Amazon S3 for science grids: a viable solution? In: DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing, pp. 55–64. ACM, New York (2008)

Complex Multiplayer Urban Design System – Concept and Case Studies

Tomasz Jaskiewicz

Delft University of Technology, faculty of Architecture, Hyperbody department
Julianalaan 134, 2628 BL Delft, The Netherlands
t.j.jaskiewicz@tudelft.nl

Abstract. This paper explores the idea of creating a software and hardware system supporting collaborative urban planning and design. It demonstrates several working case studies of various parts of such system. Using these examples a selected strategy for a computer supported cooperative work for the field of architectural and urban design and planning is illustrated. Proposed strategy is part of the Protospace system and laboratory development at the Delft Univesity of Technology, faculty of Architecture.

1 Introduction

Urban planning is an inherently collaborative activity, where multiple decision makers, stakeholders or even general public may participate. Numerous, complexly interlinked factors influencing the plans are involved in the project process. It is not uncommon for urban plans to develop into several versions and multiple scenarios. Creation of an urban plan is never a predetermined process. Urban plans and strategies are often revised, corrected or even entirely replaced before their final implementation [1]. This is primarily due to the very high degree of complexity of urban systems, resulting in significant unpredictability of occurring activities and effects that interventions into such systems may cause [2]. It is also acknowledged that urban planning can be approached as an act of creating and/or modifying complex (adaptive) systems [3].

The state-of-the-art in design support systems in the field of urban planning is underdeveloped in comparison to architecture or other design fields. Design support provided by existing systems is very limited and facilitates only selected tasks. GIS [4] (Geographical Information Systems) applications are relatively common tools that can be employed to map various features of urban and other plans to geographic locations. Nevertheless, basic drafting software such as Autodesk Autocad or Adobe Illustrator, or even manual drafting techniques are to this day the most widely used tools for creating urban plans, typically accompanied by vast textual descriptions. All those approaches leave collaborative aspects of the design to take place without any form of computer support.

Many attempts were made to increase the employment of digital technologies in urban planning in order to facilitate dealing with the complexity of those undertakings. The first group of these attempts includes development and use of spatial decision support systems (sDSS) [5] aiming at improving the collaborative aspect of urban planning and design. The second group brings together extensions of traditional CAD (computer aided design) drafting techniques or GIS systems, by actively involving agent-based simulations, parametric urban modeling, genetic algorithms, various applications of neural networks, data mining and many other techniques. Nevertheless, no tools or standards in either category have yet been commonly accepted by the wide community of urban planners.

Case studies presented in this paper are aimed at investigating possibilities of a synergy between these two trends, while allowing further exploration of new possibilities on both tracks by pursuing an integrated approach towards creation of an extensible urban design support system. Its development is based on design system architecture formulated by the team including the author and other members of the research group Hyperbody in the scope of the ongoing Protospace [6] project.

Protospace constitutes of a software system (Protospace software) and physical environment (Protospace lab) that are being developed to facilitate collaborative designing in architectural and urban design context, using novel computer design tools and multimodal interfaces. Certain collaborative design activities of Protospace, such as design sessions involving selected specialists may, happen in the physical space of the laboratory. Nevertheless, there is large demand for facilitating online connectivity to the design model for a wider group of specialists, authorities and last but not least the general public.

2 Approach Strategy

Most commonly, urban and architectural design systems are based on hierarchical models, where hierarchies follow the scale of (usually nested) design components. Traditionally, the main canvas for such systems is established by the meta-hierarchy of a: region - city/landscape - building/street/square - interior/finishing/furniture. This approach requires revision, because in reality dependencies between design components, occurring also across different scales, are bidirectional [7] and form multiple feedback loops. For that purpose investigations were made into the concept of behavioral modeling in urban and architectural design. Behavioral modeling [8] involves creation of virtual models that are composed of multiple autonomously operating components, dynamically related to each other. In this way models can be created that: a) exhibit dynamic properties b) may natively include agent-based systems c) their different components can be modified or manipulated simultaneously, without concern for the hierarchical dependencies d) are open for further extensions, also throughout a particular design or planning process. At the same time, however, such strategy requires substantial amount of computing power, exponentially increasing with the amount of elements and users involved in the design process.

In this context, behavioral modeling needs a larger system to be embedded in. Due to the nature of urban design and planning, such models need to be formulated collaboratively and need to be connected to other models, depending on the specificity of the project. In that process, ideally all participants would be present at the same location. In reality this is often not viable and, therefore, a wide range of possibilities for remote on-line access to the design model is required.

As a working concept, it has been agreed that urban design system should support the entire process of collaborative and multidisciplinary design. It should allow for parallel work on the main design model, real-time connectivity to other models, and it should provide a possibility of creation, simulation and validation of multiple design variations. The design model should be open for extensive modifications and adjustments throughout the entire design process and after project completion. In addition to that, it was decided that all urban activities should be performed in a real-time navigable 3d environment instead of a two-dimensional one, the latter being still commonly practiced in urban design and planning.

3 System Vision

Proposed solution model for the problem consists of three generic layers. The server(s) host all project data. Various tools (client applications) connect to project database. Access rights vary among them and depend on their role in the system. Client applications may include web applications, dedicated specialist tools or collaborative design support systems dealing with multiple user teams. The third layer includes a flexible set of interfaces that can be used to control client applications. Typical mouse/keyboard/screen interface can be replaced by CAVE (cave automatic virtual environment) systems, alternative pointer devices, gesture and speech recognition and other interfaces facilitating work in a collaborative environment.

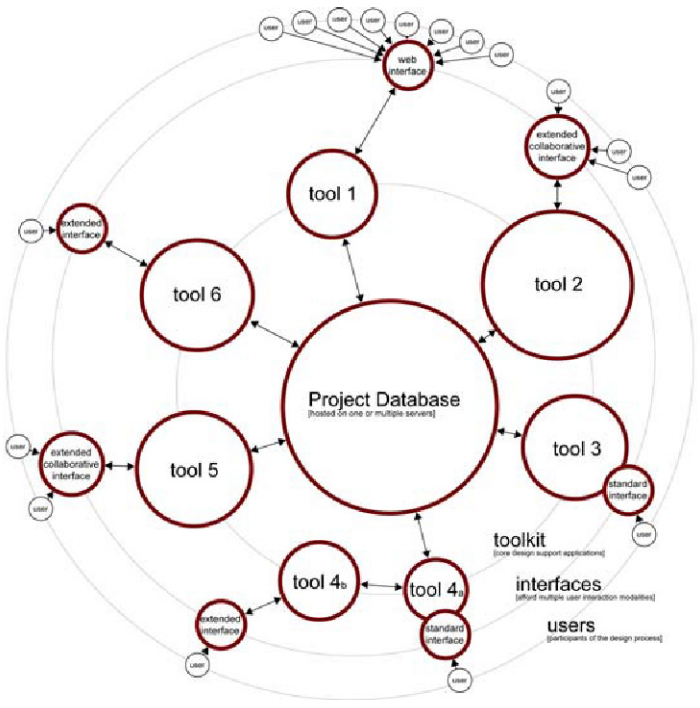

web interface
extended collaborative interface
extended interface
tool 1
tool 2
tool 3
tool 4b
tool 4a
tool 5
tool 6
Project Database
[hosted on one or multiple servers]
standard interface
extended collaborative interface
user
toolkit
[core design support applications]
interfaces
[afford multiple user interaction modalities]
users
[participants of the design process]

Fig. 1. Complete range of proposed system architecture variations for creating a multiuser design system model. Branches of this system were independently explored in case studies presented further.

Although there are no analogous systems in existence, client applications available to members of the general public willing to participate in the urban design process or obtaining information about developed plans could be compared to features currently available in version 5.0 of the Google Earth [9] service. In a similar way in which 3d building models can be uploaded and visualized in that application, variants of buildings and other future development plans could be displayed in the client application of the proposed system, allowing general public to view, express opinions, debate or vote for most desirable solutions (optionally using real-time controlled avatars). On the other end of the spectrum of possibilities, existing specialized applications such as road network design or traffic simulation software (e.g. OmniTrans International) could be connected to the system through custom APIs (application programming interfaces).

4 Case Studies

On the path of agile development and identification of the specificity of problems that are expected to be faced when creating the proposed system, several research projects and case studies were conducted at the Delft University of Technology, faculty of Architecture, the Hyperbody group, under the umbrella of the Protospace laboratory development, in which the author has actively participated. Several other projects by the author that are also presented in this paper are satellite projects of this development, carried out in connection to educational activities of Hyperbody group.

All explorations were strongly rooted in ideas influenced by computer games. Most important concepts were this of a multiplayer game system and real-time interaction in a 3d environment. The main structure of proposed systems involved a server application (MySQL based, or custom Virtools Dev game server solution) and several diverse client applications or multiple instances of one application. Virtools Dev (currently 3DVIA Virtools), visual programming environment for prototyping 3d computer games, was used as a development platform.

4.1 Paracity Project

The Paracity [10] project continued on the ideas developed in an earlier project: Protospace Demo 1.1 case study. Protospace Demo 1.1 was developed by the author in a team directed by prof. Kas Oosterhuis, involving cooperation of dr. Nimish Biloria and Dieter Vandoren. It was meant as a conceptual prototype study for an architectural design support system. The development was based on the concept that a structural engineer, project manager, architect and material expert would work together on one design model, having each a different interface to it. Thus, the architect would insert and deform functional volumes. The structural engineer would control the structural topology, individual lengths and sections of structural struts. The material expert would choose materials and control sub-surface deformations and the project manager would work with a spreadsheet overview of all materials and involved costs. Due to short project timeframe, the four views were switched in a sequence, yet it would have been possible to provide them simultaneously. The software included an internal project database that was simultaneously accessible by different client applications.

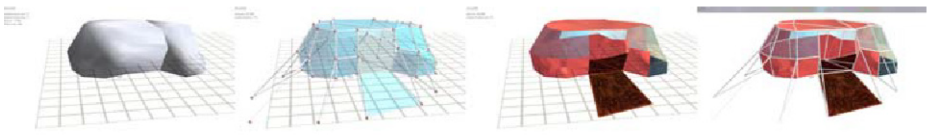


Fig. 2. Screenshots of architect, structural engineer and material expert interfaces of Protospace Demo 1.1 application, followed by the real-time render of a combined model

The later Paracity project was built on a similar principle, yet its aim was to support urban scale design. However, the application did not consist of several interfaces to the same tool, but went beyond that concept by creating a distributed system of multiple autonomous applications, each corresponding to a different aspect of the proposed design process.

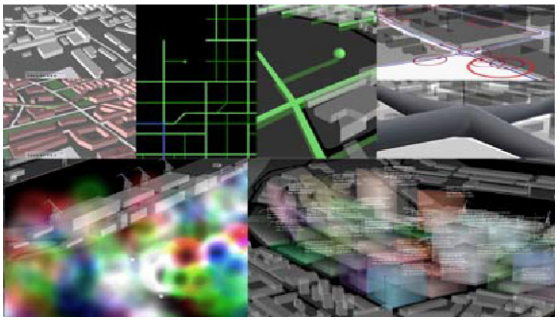


Fig. 3. Screenshot collage of all consecutive client applications of the Paracity project

The prototyped design system was consisted of seven “layers”, each layer operating as an autonomous application. First layer was providing a context of the project by defining the state of neighboring areas. The second layer was establishing the topology of all spatial connections within and around the designed area, including roads and pedestrian pathways. The third layer was evaluating that topology based on program distribution and connectivity, using an agent-based simulation. The fourth layer was calculating the potential intensities of various types of user movements through the project area and estimating probabilities of different types of functions to spontaneously appear in an urban environment. Based on those probabilities a program distribution was generated in the fifth layer, whereas the sixth layer allowed for top-down insertion of space organizing points and lines that interactively modified the generated layout in three dimensions. The last, seventh layer, dynamically generated a half-abstract 3d project model. The model is a representation of all parameters constituting the plan, including specifications of functional volumetric masses, urban block envelopes and program distribution shown as color gradients and numeric data. Streets and urban topology are shown schematically as remaining spaces between the blocks.

Unlike in other computer aided approaches to urban design, all operations in the system were performed in real-time, at more than 15 iterations (including rendering) per second. Similarly to the Protospace Demo 1.1 project, the Paracity application did not offer any possibilities of multiuser operation, other than sequential switching between different layers of the process that were meant to be operated by different design process participants, however that functionality could have been potentially added to the system if the project had been developed further.

4.2 Protospace Demo 1.2

In terms of content, Protospace Demo 1.2 project is analogous to the 6th layer of the Paracity project and can be considered to be a detailed elaboration on the activity of program distribution during the urban design/planning process. The project team included same participants as Protospace Demo 1.1, with additional support from dr. Bert Bongers and Maaslab office specializing in interaction design. The main purpose of the project was to test a possibility of multiple user interactions with the design system using one display and multiple controllers. For this, a large projection display was used in combination with two wireless game controllers, a standard OSX speech recognition system, pressure sensor embedded floor mat and an IR beacon tracking system computed using the MAX|MSP platform. The system was designed to be operated by a team of four design process actors. Each of the team members was controlling a different cursor. An algorithm was developed to control camera position and direction based on the location of cursors, causing the virtual camera to move backwards if the cursors were far from the center and forward if the cursors were in the inner part of the screen. Additionally, the view would rotate if an average position of all cursors was close to one of the screen edges. In this way 3d navigation was performed intuitively and collaboratively.

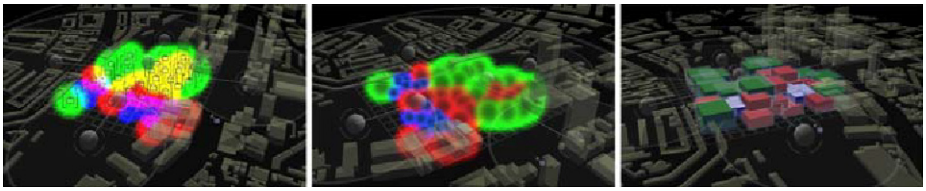


Fig. 4. Screenshots of the running Protospace Demo 1.2 application

Each of the team members used a different aspect of the interface. The urban designer would insert space organizing objects into the system by guiding the cursor using a game controller. The controller was additionally equipped with an IR beacon, allowing for 3d tracking of its position. In this way selected elements could be manipulated not only by controller sticks and buttons, but also by movements of the entire device. Similarly, the other team member, the planner, would manipulate particles representing specific parts of the functional program, drag them into desired areas and allow them to find the most appropriate location in that area on their own, based on their pre-programmed behavior. The cost expert would operate in a radically different way, by walking around the pressure-sensitive floor, which represented the project area. In this way his position in real space was mapped to the virtual environment. Based on that input, location specific information was displayed and cost calculations for chosen areas could be adjusted. The last team member, the session leader, was only using a headset to switch between different phases of the process and save states of the model, while freely moving around, discussing with other team members and supervising the progress of their work.

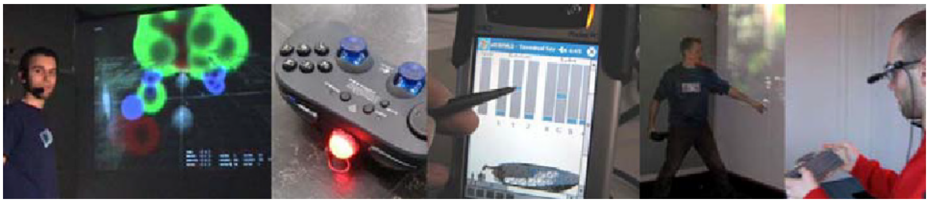


Fig. 5. Interfaces of the Protospace Demo 1.2 project

The resulting installation verified the possibility of having a small team of specialists working collaboratively on one model from the same location. The concept proved successful. However, four team members was the maximum number that was feasible without significantly decreasing team productivity. Further experiments on interfaces included use of PDA-based input, custom built controllers and augmented displays for role-specific data display.

4.3 751 Project

The 751 project was supervised by prof. Kas Oosterhuis and executed by the author as part of the Hypebrody Master of Science design course. It was an attempt to validate a possibility of a very large team of designers working on one urban design project. The approach covered a different problem than the previously described projects. In this case all design group members had the same role of urban designers, but were assigned to different zones of the design site. The provided site and given assignment were set in a way to force designers to be strongly dependent on each other's decisions. To motivate them to work in an out-of-the box manner, without any design method preconceptions, the site was defined as a three-dimensional volume in place of traditional two-dimensional plot. Design sites were also distributed in three dimensions. This meant that some of the designs had to be located above or under other, often without any direct access to the outside boundary of the design area. In this way projects were mutually dependent, forcing designers to collaboratively solve problems of accessibility, structural support, light access, connectivity, transportation and many other.

To facilitate that functionality, a database system was developed for the use of the project, which, in this case, was not a usual repository of project parameters, but was dedicated to managing only the data being exchanged between projects. Every surface separating any two adjacent zones was mapped to a different table in the database. Each record consisted of 2d position coordinates on that surface, a 3d vector of direction of occurring exchange, its value, units and most importantly, category of exchanged information. In this way it was left open to designers to decide what information was to exchange in the system. Both flows of people and structural forces could be equally well expressed using this data model and it was up to individual designers to decide how this data were to be interpreted in their individual projects. Originally, the communication with the database was performed using a standard online interface. Throughout the duration of the project an interactive application was developed that worked as a viewer of the entire site and allowed for more intuitive selection and definition of information to be exchanged.

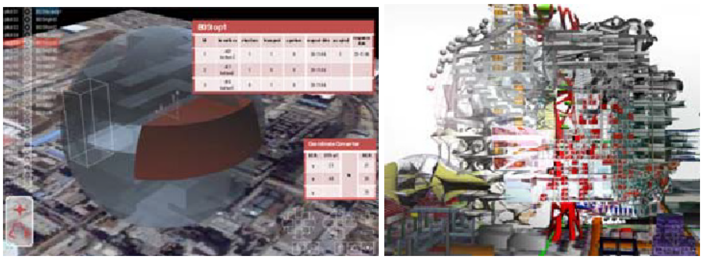


Fig. 6. 751 project, from the left: partially implemented custom database interface application and the assembled model in a real-time viewer application

Workflow progress was twofold. It consisted of working from remote locations and personal meetings in subgroups, rarely involving the entire group, which included 23 designers in total.

The process of development of the project in many ways resembled the growth of real city structures. Since exchanged parameters were forming multiple feedback loops, individual sub-projects were continuously being reconfigured and after five months of the duration of the project, no complete equilibrium was established. However, throughout the process, the project as a whole has evolved into a rich, interesting and potentially well functioning city-scale structure of an unprecedented scale and form. Despite a certain degree of design task abstraction, it was proven that complex designs can be created in an entirely distributed manner, since the used database was storing only locally exchanged parameters.

4.4 A2 Design Studio

The A2 design studio was taught by dr. Nimish Biloria with H.C.Friedrich and the author assisting and developing the design support systems. The system developed for the A2 design studio was in certain aspects similar to the 751 project. Yet, it involved a more realistic design assignment. In this case the individual zones were areas along the A2 highway in the Netherlands, not forming a 3d structure, but a more conventional two dimensional plan. In this case each zone had only two neighboring plots on its two sides and a straight line as a connection. The collaborative design support system was created in a similar manner, yet this time more attention was given to the software prototype, including the functional distribution behavioral modeler which became the core part of the project.

The application was embedded in an on-line website that upon opening, requested the plot number to be edited and in this way allowed for working on all design zones simultaneously, from any location where a standard PC computer and an internet connection were available. Additionally, the system included another client application, which was overlooking the entire project development in a top-down manner, establishing global parameters, accessible from all different zones. These parameters were: overall program values for specific functions to be distributed throughout the whole design site and included guidelines for building heights along the highway represented by a three dimensional curve. In this way each point on the ground plane in the entire site was mapped to a specific preferred height value.

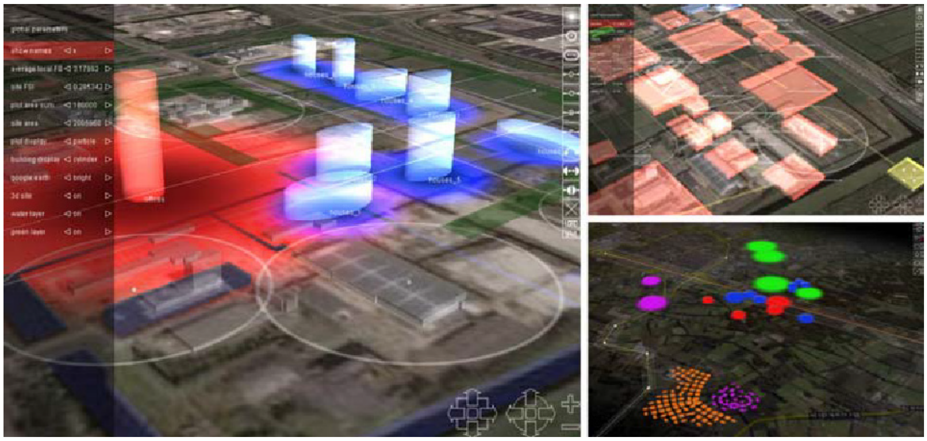


Fig. 7. Screenshots of the A2 application for distributing functional program elements

In order to simulate program demand distribution occurring in real life economy, insertion of each element of the functional program was causing an increase of demand for other functions. This demand was spreading to adjacent zones, while its value would also exponentially decrease. A special matrix model was used to calculate demands for different functions.

5 Conclusion

Presented case studies explore different components of what could together form a fully operating multiplayer urban design and planning system. Most of verified technical possibilities are being currently applied to the Protospace system framework, which in the near future will serve as a platform for implementing proposed solutions in their full potential and validate them by testing the system on applied projects executed in collaboration with the commercial sector.

The outcome of presented case studies has been highly influential on the development of the Protospace System. An additional developed feature of that system, which has not been explicitly demonstrated in presented examples, is integration of client applications of external developers. In shown projects, multiple "views" on the design are created, either by specialized interfaces, or by connecting multiple applications to one project database. For this, presented systems, such as in the Paracity project, consist of different specialized modules (client applications). In reality, these modules need to perform much more complex tasks than in presented examples and it may not be feasible to develop them within one research group or small company. On the other hand, many commonly used commercial applications allow scriptable connections to external servers and/or connectivity to their APIs. In this way those applications could replace selected modules of proposed systems. Several solutions were tested and results are promising.

However, amount of data being exchanged increases as the system approaches real-life applications and with each additional module being introduced. For this H.C. Friedrich has introduced a XiGraph [11] data structure and protocol concept, which has the potential to become an additional layer of the system, mapping and controlling all connected parameters, allowing for multiple databases to be integrated along with a possibility of flexible connectivity between modules, evolving throughout the project. XiGraph protocol supports flexible creation of dynamic connections and dependencies between data sets as well as additionally solves potential conflicts between applications when simultaneously editing same project data.

The development of Protospace system has been put on hold as a result of the disastrous fire of the Faculty of Architecture in Delft, on May 2008. The project has been recently resumed with new hopes for further developments.

References

1. Hopkins, L.D.: Urban Development: The Logic of Making Plans, 1st edn. Island Press (2001)
2. Batty, M.: Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals. The MIT Press, Cambridge (2007)
3. Portugali, J.: Complex Artificial Environments: Simulation, Cognition and VR in the Study and Planning of Cities, 1st edn. Springer, Heidelberg (2006)
4. Wilson, J., Stewart Fotheringham, A.: The Handbook of Geographic Information Science. Wiley-Blackwell, Chichester (2007)
5. Power, D.J.: Brief History of Decision Support Systems (DSSResources.COM, World Wide Web), http://DSSResources.COM/history/dsshistory.html, version 4.0 (March 10, 2007)
6. Oosterhuis, K., et al.: Protospace Software. In: Second International Conference World of Construction Project Management, presented at the Second International Conference World of Construction Project Management. TU Delft, Delft (2007). http://www.wcpm2007.nl/
7. Kilian, A.: Design Exploration through Bidirectional Modeling of Constraints. MIT Press, Cambridge (2006), http://www.designexplorer.net/newscreens/phd2006/index.html
8. Reynolds, C.: Flocks, Herds, and Schools: A Distributed Behavioral Model. Computer Graphics 21(4); SIGGRAPH '87 Conference Proceedings (1987)
9. Google Earth, http://earth.google.com/
10. Jaskiewicz, T.: Paracity – A Digital Urban Design Process. In: Game Set And Match II. On Computer Games, Advanced Geometries, and Digital Technologies. Episode Publishers, Delft (2006)
11. Oosterhuis, et al.: Protospace Software

UNICORE Summit 2009

Preface

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" from October 11 to 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, from August 30 to 31, 2006. The proceedings are available as LNCS 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 4854. The UNICORE Summit 2008 was held in conjunction with the Euro-Par 2008 conference in Las Palmas de Gran Canaria, Spain, on the 26th of August. The proceedings are available as LNCS 5415. In 2009 the 5th UNICORE Summit was held again in conjunction with the Euro-Par conference, this time in Delft, The Netherlands, on the 25th of August.

We would like to thank the Program Committee members Agnes Ansari, Rosa Badia, Donal Fellows, Anton Frank, Edgar Gabriel, Alfred Geiger, Erwin Laure, 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 Roger Menday for providing additional reviews.

The papers presented at the 5th UNICORE Summit were focussing on the areas: extensions of UNICORE, data management and deployment. The quality of the submissions was high, thus the acceptance rate was 66%.

Finally, we would like to thank all authors for their submissions, camera-ready versions, and presentations at the UNICORE Summit 2009 in Delft as well as Morris Riedel for giving the opening talk.

More information about the UNICORE Summit series can be found at
`http://www.unicore.eu/summit`

We are looking forward to the next UNICORE Summit !

September 2009

Achim Streit
Wolfgang Ziegler

JavaGAT Adaptor for UNICORE 6 – Development and Evaluation in the Project AeroGrid

Anastasia Eifer[1], Alexander Beck-Ratzka[2], and Andreas Schreiber[1]

[1] Simulation and Software Technology
German Aerospace Center
51147 Cologne, Germany
`Anastasia.Eifer@dlr.de, Andreas.Schreiber@dlr.de`
`http://www.dlr.de/sc`

[2] Max Planck Institute for Gravitational Physics
Potsdam, Germany
`alexander.beck-ratzka@aei.mpg.de`
`http://www.aei.mpg.de`

Abstract. Making applications *Grid-aware* (or *Grid-enabled*) requires the knowledge of the API of different Grid middlewares. The complexity of these systems, both in terms of underlying technology and functionality, remains high. Moreover, Grid middlewares like Globus and UNICORE are still undergoing many changes. Grid Application Toolkit (GAT), a high-level API for accessing Grid services, provides application developers with a unified, simple and middleware-independent interface to the Grid.

In this paper, we present a newly developed adaptor for GAT to access UNICORE services. We also describe how the data management client DataFinder has been Grid-enabled with GAT to access remote files and submit computational jobs to the Grid. DataFinder provides primarily an easy-to-use tool for scientific data management in distributed environments. Within the project AeroGrid, a BMBF-funded project in the German D-Grid initiative, DataFinder is being used as user interface for performing complex simulations in a UNICORE-based Grid infrastructure.

Keywords: Grid Application, UNICORE, GAT, DataFinder.

1 Introduction

The AeroGrid project [1] provides an efficient Grid based working environment for the aerospace research community. The AeroGrid environment will be a permanent and effective Grid infrastructure for the cooperation between industry, research centres, and universities in aerospace engineering and research. In this context, an important role is played by large simulation codes that are developed in several institutes of the German Aerospace Center (DLR) and at universities.

Since these codes use very innovative algorithms they are more powerful than many commercially available products. The simulation codes are used either directly by industrial companies or by research partners on their behalf. Examples are the TRACE turbine flow solver [2] of the DLR Institute for Propulsion Technology, or the TAU unstructured hybrid flow solver [3] of the DLR Institute for Aerodynamics and Flow Technology. These codes are employed both industrially and in university training and research.

The basic Grid middleware used for the AeroGrid infrastructure is UNICORE 6 (*Uniform Interface to Computing Resources*) [4,5]. The AeroGrid environment consists of two different user interfaces, a Web portal and the data management client application DataFinder [6]. DataFinder is a lightweight application software for managing technical and scientific data [7]. This tool has already been adapted to use Grid storage resources in the D-Grid [8] Integration Project (DGI-1). The main functionalities of DataFinder are data organization, annotation of standardized and user-defined metadata on data objects, provision of a metadata-based search, and script processing to automate workflows (e.g., automatic up- and downloading or calculations). Within the project AeroGrid, DataFinder is being extended with interfaces to UNICORE 6 and being used as user interface for performing complex simulations.

The primary aim of the *Grid Application Toolkit (GAT)* [9,10], developed by the EU-funded GridLab project [11], is to decouple the application from the available Grid middleware and its services. It allows developers to easily include Grid functionality in application codes by providing them with a uniform interface to numerous types of Grid middleware. This is performed by writing plug-ins (called adaptors) for the different Grid middlewares. With the help of GAT DataFinder can access a wide range of Grid services and middlewares, in a transparent, secure and effective way, without installing middleware on the submitting (client) host.

The paper is organized as follows. In Section 2 we introduce GAT, the background technology for our work. Section 3 describes the concept and implementation of the UNICORE adaptor (middleware binding) for GAT. Section 4 presents the DataFinder job management design and implementation in detail and illustrates its current use. Finally, conclusions are drawn in Section 5.

2 Background

Since our work is based on the Grid Application Toolkit, we briefly introduce this technology.

2.1 Grid Application Toolkit

GAT represents a unique and easy Application Programming Interface (API) to access the Grid irrespective of the middleware which finally triggers the Grid operation. So a Grid operation only requires the knowledge of the GAT–API, and not the knowledge of the API of different Grid middlewares like UNICORE, Globus Toolkit [12] or gLite [13]. Programming against the GAT–API is much easier than coding against Grid middleware directly. According to our experience, a job submission using the Globus API requires around 100 lines of code, the same job submission using GAT only 20 lines. The same relation is valid for file operations.

2.2 GAT Architecture and Implementations

The GAT architecture is divided to two parts: *GAT engine* and *GAT adaptors*. The GAT–API delivers the commands to the GAT engine which selects a so–called GAT adaptor. Such an adaptor converts a file copy or job submission statement written in the GAT–API into a file copy or job submission statement of a Grid middleware. GAT uses the first adaptor which does not fail but it is also possible to force the usage of selected adaptors.

GAT has first been realized in **C** within the European GridLab project; the development of the Java version of GAT (henceforth JavaGAT) has also been started during the GridLab project. The **C** Implementation comes with a **C++** and a Python wrapper. While C–GAT is out of support, JavaGAT is still maintained by the Free University of Amsterdam, and JavaGAT is used within AeroGrid. One of the major advantages of JavaGAT compared to C–GAT is that JavaGAT enables the Grid access without an additional installation of a Grid middleware. All required software to create a Grid client is rolled out with a JavaGAT installation which does not take more than 5 minutes, compared to usually several hours for a Globus installation.

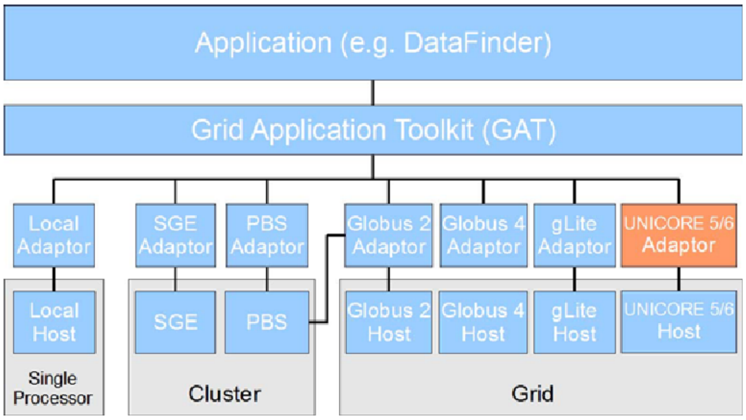


Fig. 1. JavaGAT software architecture

Figure 1 shows the JavaGAT architecture. Beside real Grid adaptors JavaGAT also offers adaptors for the Sun Grid Engine (SGE), Portable Batch System (PBS) or simply for local host operations. The availability of local adaptors enables the software engineer to create his code without needing access to the

Grid. After the program logic is working correctly on the local host, the Grid can be taken into account. The recently developed UNICORE adaptor for GAT is marked red in Figure 1 and will be described in Section 3. JavaGAT is freely available as open source software.

Beside AeroGrid, the JavaGAT API has been used by several other projects and institutions:

– AstroGrid–D [14], the German Astronomy Community Grid (GACG), uses GAT to enable the Grid access for the workflow engine ProC [15].
– TextGrid [16,17], one of the first projects in the humanities in Germany and Europe, uses GAT to read and write TextGrid objects.
– PartnerGrid [18] uses GAT for the Grid interface in Reconfigurable Computing Environment (RCE) [19].
– Vrije Universiteit Medical Center in Amsterdam (VUMC) submits complex medical applications to the Grid and performs also its data management using GAT.
– Amsterdam Medical Center (AMC) uses GAT for Medical data management.
– AMOLF, the Institute for Atomic and Molecular Physics in Amsterdam, uses GAT in a Fourier Transform Mass Spectrometry (FTMS) analysis application. The FTMS dataset can be streamed to compute resources with GAT using SSH, SFTP and GridFTP.
– The workflow system Triana [20] uses GAT to start jobs on the Grid.
– GAT will complement the Ibis functionality, a Java-based grid programming environment [21].
– Technical University of Catalonia (UPC) and Barcelona Supercomputing Center (BSC) use GAT for job submission and file transfer in the COMP Superscalar Grid framework [22].

Furthermore BWGrid is considering to use JavaGAT, and also the High Performance Computing Center Stuttgart (HLRS) of the University of Stuttgart is planning to use GAT for the job submission to PBS clusters or UNICORE resources within the context of the LarKC project [23].

Simple API for Grid Application (SAGA) [24], now being developed by a research group within the Open Grid Forum (OGF), can be understood as the successor of GAT. The primary goal of SAGA is to develop a standardized, simple Grid API. JavaGAT itself is included in the Java implementation of SAGA from the Vrije Universiteit (VU) in Amsterdam which makes all JavaGAT–supported middleware (e.g., Globus, gLite, UNICORE, or SSH) available for SAGA.

3 UNICORE Adaptor for JavaGAT

The UNICORE adaptor has been implemented recently at the Max–Plank–Institute for Gravitational Physics within the scope of the DGI2–FG1 project of D–Grid [8]. The implementation is based on HiLA [25]. HiLA (High-Level API) supports the access to UNICORE 5 and UNICORE 6 via an easy and unique API. Furthermore it is not necessary to install components of UNICORE 5 or UNICORE 6 on the submitting (client) host.

The submission of jobs within the UNICORE adaptor of JavaGAT consists of the following steps [26]:

1. HiLA requires the existence of a so–called `site` object, which can only be created if a valid and active execution site (or host) for UNICORE jobs exists.
2. HiLA submits a job only with a JSDL based job description (*Job Submission Description Language*). This JSDL description requires the name of the executable, and its arguments. The UNICORE adaptor of JavaGAT creates the JSDL description with information of the GAT `JobDescription` object; a `JobDescription` is necessary to submit a job with GAT.

   Pre- and post-staging is not handled in the JDSL description of a HiLA job. These operations are done separately by using the methods `importLocalFile` or `exportToLocalFile`, of the HiLA `File` class. The informations about the files which are to be pre-staged or post-staged are available in the GAT `JobDescription` object. The pre-staging is done by the UNICORE adaptor before starting the job, the post-staging after it has been finished.
3. Having the JSDL description, one can create a job task using the `submit` method on the HiLA `site` and the "submission" returns a `task` object which must be used for further operations on the job. The following operations are implemented up to now:
   – starting the job;
   – retrieving of the job status;
   – stopping the job;
   – receiving the exit status of the finished job.

   For all these operations the UNICORE adaptor of JavaGAT provides methods, which translate the statements of the GAT–API into statements of the HiLA–API.

4 DataFinder Job Management

4.1 Design Approach

In this section, we describe the design for the DataFinder job submission and management system. The DataFinder job management includes the basic functionality required to manage a job: resource discovery, authentication, job submission, file transfer, and job status notification. Furthermore, the users are allowed to resubmit and archive old jobs, and visualize job input and output with pre- and post-processing tools.

The DataFinder design supports a three step process to job execution (job life-cycle): *job creation*, *job submission* and *job monitoring*. During the first step the job is constructed, defined by a job object with the correct host, backend and application parameters. Afterwards it will be submitted to the target host, using proper authentication and file transfers. Finally, the job status is retrieved. If the job completed successfully, the output files will be copied to the local host.

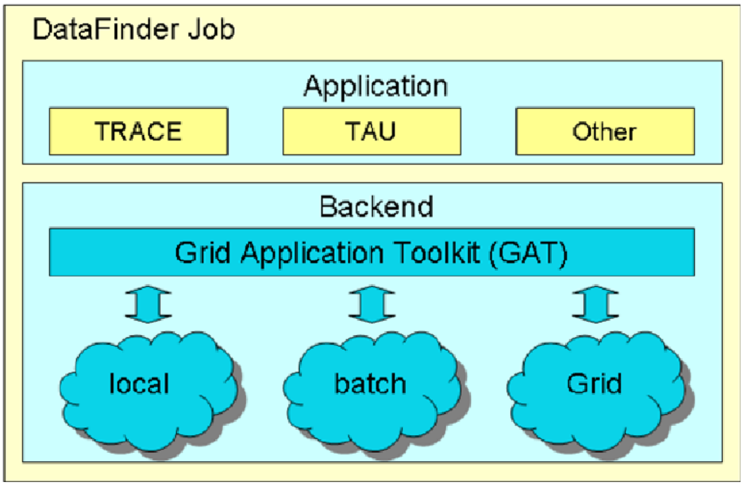


Fig. 2. Basic components of the DataFinder job management system

Figure 2 shows an overview of the modular design of the DataFinder job management. The basic job management components include:

– Job (job definition component);
– Application (application definition component) and
– Backend (resource definition component).

The Job component is represented as an object that includes all job characteristics and requirements such as the current state of the job, a job working directory, number of CPUs, start time, end time, and batch queue. Each job (or instance of a job class) is identified by the job's identifier, specified when the job was created (unique DataFinder job identifier).

The Application component defines all application-specific information such as executable (a single executable binary or a single shell script), program arguments and environment settings, all staging files from the local machine and all result files on the computing host.

The Backend component represents a resource or a batch system and includes information such as hostname or middleware type. DataFinder supports a number of backend plug-ins which allow the execution of jobs locally and in a distributed environment. The specialized backend components such as *Job Handler* and *File Handler* implement interfaces for different types of batch and local systems. These handlers have complete freedom to implement their own mechanisms for the local, Grid, and batch submission, so one can write new handlers and choose between the existing ones. In particular, the Job Handler performs the job configuration and submission to a chosen processing system and monitors the job progress. The File Handler transfers input and output files between a local and a remote site.

4.2 Implementation

DataFinder is implemented in Python [27], an interpreted scripting language, using an object-oriented approach. In order to access Grid resources, we used the Java version of GAT. It has all required functionality, ist widely used (Section 2) and a UNICORE Adaptor for C-GAT (and for its Python Wrapper) is not available.

Bridging the two languages Python and Java ist not trivial, but there are different techniques and libraries to access Java from Python like JPype [28] or JCC [29]. JPype is an open-source Python library that allows Python programs full access to existing Java class libraries. This is achieved by embedding a Java Virtual Machine (JVM) instance within the host's Python process. JCC is a C++ code generator that produces a **C++** object interface wrapping a Java library via the Java Native Interface (JNI) [30].

JPype and JCC both allow the usage of Java libraries from within Python code. We found JPype to be a better solution by reason that JCC requires a separated code generation and compilation step (no "on the fly" wrapping). DataFinder requests JPype to initialize a Java Virtual Machine, which uses JNI to communicate back and forth (Figure 3). A Python application code (DataFinder backend component) then imports and uses JavaGAT classes available from the JVM. In order to submit a job the Job Handler calls the Java-GAT API and wraps the Python calls which are then forwarded to the JVM by JavaGAT with the help of JPype.

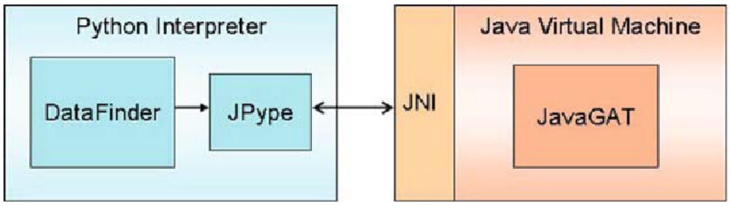
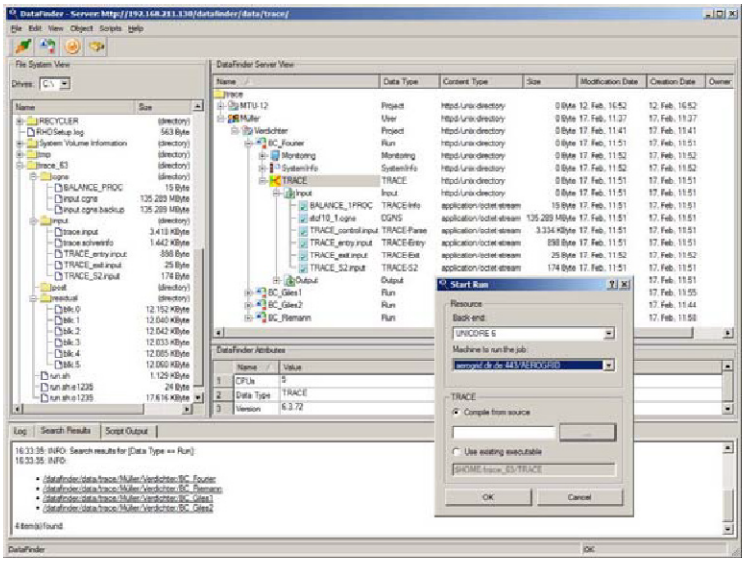


Fig. 3. Accessing JavaGAT libraries from DataFinder via JPype

4.3 Job Submission through DataFinder

The process of job submission through DataFinder is shown in Figure 4. The main DataFinder GUI window is divided in two parts: The left section shows a local file system view; the right section is used to display a server view. With the help of the "Create Run" dialog the user creates a new job, specifying the application type and required input files. Now the user can call the "Start Run" option where he can select the resource type (local host, remote host, batch, grid) and location (hostname or ip address) on which the job will be run. DataFinder connects to the chosen resource, transfers the required input files and executable of the job to the resource, and starts the job.



Name	Size
RECYCLER	(directory)
RHDSetup.log	563 Byte
System Volume Information	(directory)
tmp	(directory)
trace_63	(directory)
cgns	(directory)
BALANCE_PROC	15 Byte
input.cgns	135.289 MByte
input.cgns.backup	135.289 MByte
input	(directory)
trace.input	3.418 KByte
trace.solverinfo	1.442 KByte
TRACE_entry.input	898 Byte
TRACE_exit.input	25 Byte
TRACE_S2.input	174 Byte
post	(directory)
residual	(directory)
blk.0	12.152 KByte
blk.1	12.040 KByte
blk.2	12.042 KByte
blk.3	12.033 KByte
blk.4	12.085 KByte
blk.5	12.060 KByte
run.sh	1.129 KByte
run.sh.e1235	24 Byte
run.sh.o1235	17.616 KByte

Name	Data Type	Content Type	Size	Modification Date	Creation Date	Owner
trace						
MTU-12	Project	httpd/unix-directory	0 Byte	12. Feb., 16:52	12. Feb., 16:52	
Müller	User	httpd/unix-directory	0 Byte	17. Feb., 11:37	17. Feb., 11:37	
Verdichter	Project	httpd/unix-directory	0 Byte	17. Feb., 11:41	17. Feb., 11:41	
BC_Fourier	Run	httpd/unix-directory	0 Byte	17. Feb., 11:51	17. Feb., 11:51	
Monitoring	Monitoring	httpd/unix-directory	0 Byte	17. Feb., 11:52	17. Feb., 11:52	
SystemInfo	SystemInfo	httpd/unix-directory	0 Byte	17. Feb., 11:52	17. Feb., 11:52	
TRACE	TRACE	httpd/unix-directory	0 Byte	17. Feb., 11:51	17. Feb., 11:51	
Input	Input	httpd/unix-directory	0 Byte	17. Feb., 11:51	17. Feb., 11:51	
BALANCE_1PROC	TRACE-Info	application/octet-stream	15 Byte	17. Feb., 11:51	17. Feb., 11:51	
atcf10_1.cgns	CGNS	application/octet-stream	135.289 MByte	17. Feb., 11:51	17. Feb., 11:51	
TRACE_control.input	TRACE-Parse	application/octet-stream	3.334 KByte	17. Feb., 11:51	17. Feb., 11:51	
TRACE_entry.input	TRACE-Entry	application/octet-stream	898 Byte	17. Feb., 11:51	17. Feb., 11:51	
TRACE_exit.input	TRACE-Exit	application/octet-stream	25 Byte	17. Feb., 11:52	17. Feb., 11:52	
TRACE_S2.input	TRACE-S2	application/octet-stream	174 Byte	17. Feb., 11:51	17. Feb., 11:51	
Output	Output				17. Feb., 11:51	
BC_Giles1	Run				17. Feb., 11:55	
BC_Giles2	Run				17. Feb., 11:44	
BC_Riemann	Run				17. Feb., 11:58	

	Name	Value
1	CPUs	5
2	Data Type	TRACE
3	Version	6.3.72

Resource
Back-end: UNICORE 6
Machine to run the job: aerogrid.dlr.de:443/AEROGRID

TRACE
(●) Compile from source
( ) Use existing executable
$HOME/trace_63/TRACE

```
16:33:35: INFO: Search results for [Data Type == Run]:
16:33:35: INFO:

  • /datafinder/data/trace/Müller/Verdichter/BC_Fourier
  • /datafinder/data/trace/Müller/Verdichter/BC_Riemann
  • /datafinder/data/trace/Müller/Verdichter/BC_Giles1
  • /datafinder/data/trace/Müller/Verdichter/BC_Giles2

4 item(s) found
```

Fig. 4. Job Submission through DataFinder

Once the job is submitted, DataFinder periodically monitors its status. All jobs begin in the *new* state. As soon as a job is successfully submitted by DataFinder, it is assigned to the *running* state. This state indicates that the input data was successfully transfered to the remote host. After the application successfully executed, the job moves to the *finished* state or, in case of failure, to the *failed* state. The output is automatically retrieved from the worker node when the job is finished. Furthermore, the user can resubmit the job to different backends and the restarted job will go back to the *new* state.

5 Conclusions

Grid APIs such as GAT and SAGA have been developed to allow easy development of Grid-aware (or Grid-enabled) applications. GAT/SAGA provides a common programming interface to the numerous Grid technologies that exist such as Globus, gLite etc. The adaptor based architecture of GAT/SAGA allows for easy adaption to new Grid middlewares. We have designed and implemented a UNICORE adaptor for JavaGAT to access UNICORE services. Using the GAT engine framework and our UNICORE adaptor, a Grid application developer can access UNICORE-based resources with the help of an easy and unique API and without needing to install components of UNICORE 5 or UNICORE 6 on the submitting (client) host.

The UNICORE adaptor is being used in the UNICORE 6 based AeroGrid infrastructure. AeroGrid provides a graphical user interface, a data management client called DataFinder. In this paper, we presented the DataFinder job management concept and the implementation using the UNICORE adaptor. DataFinder, by design, will use existing Grid middlewares and provide a simple and unified access to them. DataFinder has been developed in Python as an easy-to-use tool for scientific data and job management, providing a single user interface for submission to multiple backends. DataFinder job management components are implemented as independent and reusable modules. Jobs can run locally, on scheduling systems and in Grid environments. DataFinder supports the transparent submission and monitoring of computational jobs to a variety of resources. The end-users do not have to know all the technical details about Grid resources, running a job locally is not different from running it on the Grid, the details of the Grid will be hidden. Experimental results showed that this approach is user-friendly and system efficient. More information about DataFinder development can be found on its web site [6]. DataFinder is available as Open Source software under the BSD license.

Acknowledgments. This work has been supported by the German Federal Ministry for Research and Technology (BMBF) under Grant 01IG07006 (AeroGrid) and Grant 01IG07014 (DGI-2).

References

1. German Aerospace Center (DLR): AeroGrid website, `http://www.aero-grid.de`
2. Yang, H., Nuernberger, D., Kersken, H.P.: Toward excellence in turbomachinery computational fluid dynamics: A hybrid structured-unstructured reynolds-averaged navier-stokes solver. Journal of Turbomachinery 128(2), 390–402 (2006)
3. Gerhold, T.: Overview of the hybrid RANS code TAU. In: MEGAFLOW – Numerical Flow Simulation for Aircraft Design. Notes on Numerical Fluid Mechanics and Multidisciplinary Design, vol. 89, pp. 81–92. Springer, Heidelberg (2005)
4. Jülich Supercomputing Centre: UNICORE, `http://www.unicore.eu/unicore`
5. Erwin, D., Rambadt, M., Streit, A., Wieder, P.: Production-quality grid environments with UNICORE. In: Wallom, D., Kielmann, T. (eds.) Proceedings of the Workshop on Grid Applications: From Early Adopters to Mainstream Users, Global Grid Forum, pp. 10–17 (2006)
6. German Aerospace Center (DLR): DataFinder, `http://datafinder.sourceforge.net`
7. Schlauch, T., Schreiber, A.: DataFinder – a scientific data management solution. In: Ensuring the Long-Term Preservation and Value Adding to Scientific and Technical Data, PV 2007, Oberpfaffenhofen, Germany (2007)
8. D-Grid Initiative: D-Grid: The german grid initiative, `http://www.d-grid.de`
9. Allen, G., Davis, K., Dramlitsch, T., Goodale, T., Kelley, I., Lanfermann, G., Novotny, J., Radke, T., Rasul, K., Russell, M., Seidel, E., Wehrens, O.: The GridLab grid application toolkit. In: HPDC, vol. 411 (2002)
10. van Nieuwpoort, R.V., Kielmann, T., Bal, H.E.: User-friendly and reliable grid computing based on imperfect middleware. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC'07). ACM/IEEE (November 2007)
11. The GridLab Project: GridLab: A grid application toolkit and testbed, `http://www.gridlab.org`

12. 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)
13. Enabling Grids for E-Science (EGEE): EGEE gLite User‘s Guide - Overview of Glite Data Management (March 2005)
14. Enke, H., Steinmetz, M., Radke, T., Reise, A., Röblitz, T., Högqvist, M.: AstroGrid-D: Enhancing astronomic science with grid technology. In: German e-Science Conference, GES 2007 (2007)
15. The AstroGrid-D Project: The Planck Process Coordinator workflow engine, `http://www.gac-grid.de/project-products/Software/ProC/proc.pdf`
16. The TextGrid Project: TextGrid, `http://www.textgrid.de`
17. Aschenbrenner, A., Gietz, P., Küster, M.W., Ludwig, C., Neuroth, H.: TextGrid - a modular platform for collaborative textual editing. In: Proceedings of the International Workshop on Digital Library Goes e-Science (DLSci06), Alicante, Spain, pp. 27–36 (2006)
18. Fraunhofer Institute for Industrial Engineering (IAO): PartnerGrid, `http://www.partnergrid.de`
19. German Aerospace Center (DLR): Reconfigurablce Computing Environment (RCE)
20. Taylor, I., Shields, M., Wang, I., Harrison, A.: The Triana Workflow Environment: Architecture and Applications. In: Workflows for e-Science, pp. 320–339. Springer, Heidelberg (2007)
21. The Ibis Project: Ibis: Grids as promised, `http://www.cs.vu.nl/ibis`
22. Tejedor, E., Badia, R.M.: COMP Superscalar: Bringing GRID Superscalar and GCM Together. In: CCGRID, pp. 185–193. IEEE Computer Society, Los Alamitos (2008)
23. Fensel, D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Valle, E.D., Fischer, F., Huang, Z., Kiryakov, A.: Towards LarKC: A Platform for Web-Scale Reasoning. In: International Conference on Semantic Computing, pp. 524–529 (2008)
24. Goodale, T., Jha, S., Kaiser, H., Kielmann, T., Kleijer, P., Merzky, A., Shalf, J., Smith, C.: A simple API for grid applications. SAGA (2008), `http://www.ogf.org/documents/GFD.90.pdf`
25. Menday, R., Hagemeier, B.: HiLA 1.0, `http://www.unicore.eu/community/development/hila-reference.pdf`
26. Beck-Ratzka, A., Zangerl, T.: GAT beginners guide, `http://www.aei.mpg.de/~alibeck/documents/gat-usage.pdf`
27. Python: Python, `http://www.python.org`
28. Menard, S.: JPype – Java to Python integration, `http://jpype.sourceforge.net`
29. The Apache Software Foundation: JCC, `http://lucene.apache.org/pylucene/jcc`
30. Sun Microsystems, Inc.: Java native interface (JNI) specification, `http://java.sun.com/javase/6/docs/technotes/guides/jni/`

Towards a Common Authorization Infrastructure for the Grid

Krzysztof Benedyczak[1,2], Marcin Lewandowski[1], and Piotr Bała[1,2]

[1] Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Chopina 12/18, 87-100 Toruń, Poland
[2] Interdisciplinary Center for Mathematical
and Computational Modeling
Warsaw University
Pawińskiego 5a, 02-106 Warsaw, Poland

Abstract. In this paper we report results of the ongoing effort to provide a seamless authorization for the UNICORE and Globus Toolkit middlewares using the UNICORE Virtual Organizations System (UVOS). The UVOS is already well integrated with the UNICORE middleware. We have designed and created a set of native Globus Toolkit 4 modules which enable a UVOS based authorization, with a similar functionality as its UNICORE counterpart. Actually the same authorization data stored on the UVOS server can serve both middlewares simultaneously. The paper provides an overview of existing approaches to the user management problem in the Grid environment with a special emphasis on those which can be used across different grid middlewares. The paper presents the UVOS system, its features and how its adoption helps to manage users of the UNICORE and Globus 4 middlewares.

1 Introduction

The problem of user management in the Grid is deliberated from the very beginnings of the Grid concept. The fundamental ideas come from the publication [1], which models the society of grid users as *virtual organizations* (VO). Over time numerous implementations were developed including a popular Virtual Organizations Management System (VOMS) [2]. In addition to the centralized VOMS, an idea of federations started to gain interest, mostly because of the success of the Shibboleth system [3] used to authenticate and authorize web users. Nevertheless the engineered solutions still contain a number of shortcomings. Primary problem is a complicated administration, lack of a fine-grained authorization and a complex deployment procedure. As there are numerous user management systems available, one of the most important problems now is the interoperability, especially when a homogenous authorization shall be provided across heterogeneous grid middlewares.

The goal of the work reported in this paper is to create a native support in Globus Toolkit 4 for user authorization, based on their data stored in a UVOS server. UVOS server is a part of the UNICORE Virtual Organizations System [4] and it is a modern VO solution developed mostly for the UNICORE grid system. However UVOS is not tightly bound to the UNICORE as it employs service oriented architecture paradigm. It uses open Security Assertion Markup Language (SAML) 2.0 protocol for communication [5] which allows for easy integration with different grid middlewares.

The native support for the UVOS system in the two significant grid middlewares can be seen as a big step towards a full grid interoperability in the area of users authorization. It is even more important as the SAML protocol used by the UVOS is utilized in the aforementioned Shibboleth federation management system. Therefore it is logical to expect that administrators will be able to switch effortlessly between those different authorization solutions or even use them together.

The next part of the work discusses the common methods of integration of the virtual organizations with the grid systems mostly using the UNICORE and Globus Toolkit middlewares as examples. The subsequent section presents in more details the UVOS system. The main part of the work — prototype of the UVOS authorization modules for the Globus Toolkit — is described in the section 5. The paper is concluded with a summary of the achieved results and an enumeration of ideas for a future work.

2 The Grid Approach to the User Management Problem

The basic mechanism of user management in most of the grid middlewares is usage of a database placed locally to a grid site (plain text file in the Globus Toolkit, a simple SQL database in UNICORE). More advanced solutions are using one of the two common approaches: the grid-central databases of users or federations. Both allow for creation of Virtual Organizations i.e. the situation when users coming from different real organizations get access to the shared resources, often located in different administrative domains.

The global database approach is simpler in installation and deployment. However it requires that central database must allow for the administration of its content (or more strictly speaking its fragments) by multiple managers with different permissions. There is also a problem of reusing users data stored in already existing catalogs such as Lightweight Directory Access Protocol (LDAP) servers. Those problems are addressed by the federation concept, where users are always authorized by their home (or real) institution. This approach is much more scalable but also more difficult to set up: it requires a mutual trust (often formal) between all user privileges providers and resource providers. A common format and semantics of users privileges must be developed and applied.

There are two practical models of establishing users authorization data used in the grid systems: *a pull* and *a push* models which are presented in the figure 1. One should note that both models may be applied simultaneously.

In the *pull mode* a service (e.g. a grid node server) contacts the VO server to obtain the attributes of a user who tries to use it. The attributes received from the VO server can be used for an authorization, e.g. server's policy may permit only those users who possess certain attributes. The service may use received attributes to perform other tasks e.g. to map requester to a local UNIX account. Pull mode is transparent for the grid users. However it is more difficult for grid administrators to set it up: every grid site must be correctly configured to use the VO server.

In the *push mode* a user has to contact a VO server on her own and get the list of possessed attributes in a signed assertion. Later this assertion can be attached to the requests which are sent to the grid services. If the service trusts the assertion issuer then it can use the attributes for authorization. The user can usually ask the VO server for a subset of owned attributes. In such a case the user can hide a part of her identity or alter the execution (e.g. by choosing role). The push mode is more scalable in terms of server administration and easier to set up. However it requires user interaction and thus is more suitable for advanced grid users. Also for the push mode a problem with expired assertions arises.

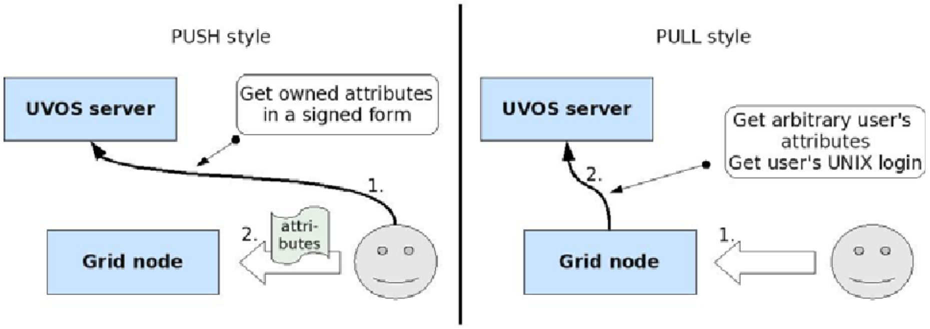


Fig. 1. The figure shows the two alternatives for provisioning client's attributes to the grid node: by the service (pull mode) and by the client (push mode)

3 Related Solutions and Efforts

One of the most important developments in the area of authorization is the GridShib project [6] based on a standard protocols. It was created for the Globus Toolkit. The primary goal of the GridShib project is creation of a full-fledged tool which integrates the Shibboleth Identity Provider as the information source about the users for the Globus middleware.

The system developed by the GridShib project is a production ready solution with a vast amount of features. It allows for using both push and pull styles of attribute retrieval. In the first case it can consume attribute assertions attached to a client's certificate (both normal X.509 and proxy). Additionally, recent releases of the GridShib software support VOMS assertions. What is interesting, the GridShib provides a modified version of a mechanism which establishes local user names for the grid clients. Instead of direct mapping of client's certificate subject name to a local account, GridShib allows for choosing the local name based on the client's attributes. The attribute to handle local name mappings are stored in a configuration file, local to the grid node.

Unfortunately GridShib can cooperate only with the old, 1.x releases of the Shibboleth system. It cannot be used with the current SAML 2.0 protocol and therefore it can not be used with the Shibboleth 2.0 or UVOS systems. Anyway we can foresee that GridShib would become the fundamental solution used for interoperable authorization of the Globus Toolkit, when it will be upgraded to the SAML 2.0.

GridShib possesses also an another shortcoming very important from the practical point of view. Namely, it is impossible to authorize clients of the legacy Globus services. Legacy services are not yet converted to the web services technology and are not OGSA [7] based. In particular there is one such a service of a critical importance: the GridFTP file transfer server. As a result GridFTP access must be based on the traditional gridmap file and there is no possibility for a complete user management via the Shibboleth federations.

In result, authorization stack for the Globus Toolkit must be implemented in two flavors: for the pre-WS and WS components. A common approach to overcome this problem is to build or even download a full gridmap file from the remote server. One of the popular tools is `edg-mkgridmap` from the Virtual Data Toolkit project [8]. Another example is GUMS software [9] which is much more advanced. Those tools, quite interesting for the Globus Toolkit are not a good base for the interoperable solution because both are using simple "user import" approach which has general problems such as low performance with large amount of users.

Along with the development of different user management systems for the grid, various efforts were undertaken to integrate them. One of the largest is the IVOM (Interoperability and Integration of VO-Management) project, a part of the D-Grid initiative. The IVOM aims to develop services that enable integration of VOMS and Shibboleth-based VO management systems with the grid middlewares used in Germany, in particular gLite, Globus Toolkit 4 and UNICORE 5. As it is explained in the publication [10] for this purposes only the push model is used. The paper suggests development of SAML enabled components for the grid middlewares which are in the project's scope. In the case of Globus Toolkit it is suggested to use the GridShib system. The paper [10] states that unfortunately GridShib does not support legacy Globus services.

4 The UVOS Overview

The UNICORE Virtual Organizations System (UVOS) is a new solution for centralized VO management. It was created in course of the EU-funded Chemomentum project [11]. The fundamental aim of the project was to develop a solution which will overcome the two important adoption blockers of a VO software: lack of flexibility and difficult deployment. The UVOS principles are:

– *Distributed environment*: The single installation can be completely controlled remotely.

- *Openness*: The system consumers can communicate with it using open and well established protocols.
- *Easy of Use*: The system can be easily installed and managed.
- *Flexibility*: The system provide tools and features which will make it useful in both OGSA (so SOA) environments and WWW environments.

UVOS architecture uses a central UVOS server which acts both as authentication service and attribute authority. The server is used by two kinds of clients: consumers and management clients. Consumers do not modify the UVOS content but query it. Management clients are used to dynamically modify VO data either by VO administrators or other management software .

The UVOS server (as the whole system) is written purely in Java so it is highly portable. All operations of the UVOS server are available via the web services interface. The server uses relational database to internally store the whole data, therefore it does not depend on any external services like LDAP.

The UVOS consumers use the open standard SAML 2.0 as a protocol to communicate with the UVOS server. The server implements the core SAML specification and to ensure a higher interoperability level it implements additional profiles:

- XACML Attribute Profile [12].
- SAML Attribute Query Deployment Profile for X.509 Subjects [13],
- SAML Attribute Self-Query Deployment Profile for X.509 Subjects [13],
- OGSA Attribute Exchange Profile Version 1.2 [14],

Finally, the UVOS web server is truly extensible by means of classic Java web applications (servlets), which can be simply installed by copying them into a designated server's installation directory. Two such web applications extending server's functionality are provided as ready to be used modules: one provides authentication form for the SAML authentication request protocol, the second one supports enrollment of new users.

UVOS access is restricted by it's own authorization stack. No external components/services are used to perform authorization. The authorization mechanism is advanced and provides a complete control of access on a group level.

4.1 UVOS VO Model

UVOS organizes VO members within a hierarchical group structure. Top level groups of this structure are called virtual organizations however are not different than other groups. Each entity can be a member of an arbitrary number of groups. It may has assigned a set of attributes. In addition, a single entity can possess multiple representations, for example in different formats. These equivalent incarnations of the same entity are called identities, and are usually invisible for an outside user.

Group membership is inherited in UVOS. The member of subgroup becomes automatically the member of the parent group. This is different than e.g. in VOMS, but has been requested by the users.

Every entity has a unique label and one or more tokens that represent it. Tokens must be in one of the supported formats, which currently are:

– a full X.509 certificate,
– an X.500 distinguished name,
– an e-mail address with an password used for authentication.

A token along with it's type is called an identity. As explained, an entity typically possesses one identity, but it can also has more. This reflects the real life situation where the single user can possess multiple certificates and email accounts. Additional identity formats may be added to the UVOS system with a intermediate level of effort. It is worth pointing out that all of the identities that compose an entity share the same characteristics (attributes, group membership, permissions, etc.): the UVOS works using entities internally.

UVOS attributes are composed of a name and a list of values. A name is a URI, and values are arbitrary strings. The value list can be empty. UVOS allows for three different ways of attributes assignment:

– global attributes: an entity can have an attribute assigned globally. Such an attribute is valid always and in every context,
– group-assigned attributes: an attribute can be assigned to a group, in which case all members of this group automatically hold this attribute (no matter if they were added later or prior to the creation of the group-assigned attribute). It is worth pointing out that this attribute is valid only in the scope of this group,
– group-scoped entity attributes: those attributes are assigned to the entity, just like global attributes, but have an additional group restriction and are valid only in in the scope of the group.

The last two methods introduce a "group-scoped validity" of attributes, which requires a further explanation. Within this mechanism the requester can ask (using the API provided by the UVOS service) for the entity's attribute either globally or valid only in a specified group. Global query returns global attributes only. A query limited to a group will return all entity's global attributes and all group-scoped attributes valid within the specified group.

4.2 The Client Side

In the UVOS currently there are available two management clients: the command line client (UVOS CLC) and VO Manager. The command line client can be used to administer UVOS from the console. It can be used in an interactive or batch mode. The UVOS VO Manager is a powerful GUI application based on the Eclipse Rich Client platform. It is much easier to use than a command line client, offers an intuitive interface so usually UVOS VO Manager a preferred choice. The VO Manager application can be considered as one of the most important advantages of the UVOS.

UVOS is available in the standard distribution of the UNICORE 6 middleware. Both push and pull modes are supported. In the push mode user can get and attach certain attributes using the UNICORE Rich Client plugin. The pull mode is implemented as the module of the UNICORE server.

5 Globus Support for UVOS

Globus Toolkit 4 provides a rich interface for building and plugging additional authorization modules. Unfortunately, as it was briefly noted above, the Globus Toolkit is currently built using two different technologies. The main part of Globus employs a web services technology and is deployed in a special Java container. The rest of services, including the important file transfer subsystem GridFTP, is programmed in C and still does not use web services technology. Those services are called as *legacy* or *pre-WS*. Obviously the authorization APIs for both types of services are absolutely different. More advanced and full of features is interface provided by a Java web services container. To make a situation even more complicated, the web services version of authorization API was greatly refactored with the introduction of the version 4.1 of the Globus Toolkit.

The issues described above may be reason for a small popularity of the pull style solutions for the Globus Toolkit. The most of the popular approaches are focused on the push model. In the case of the Globus Toolkit pushed attribute assertions are embedded inside a user's proxy certificate. This pattern is used for both VOMS credentials and it is a typical usage scenario for GridShib deployments.

In our solution the pull model was chosen as a base for the implementation. The main reason for this decision was that client tools need not to be modified in any way for pull style. This is an important factor as Globus community is quite well established and in our opinion it would be hard to convince existing users to use an another client side tools or wrappers, as GridShib does.

The outcome of our project is a set of Globus Toolkit authorization modules. The modules allow for the pull style authorization based on the information stored in the UVOS server. The modules provide this functionality for *all* Globus Toolkit 4 modules, both 4.0.x series and more recent 4.1.x and 4.2.x. There are three principal functional components of our implementation:

- UVOS Policy Information Point which solely contacts the UVOS server and collects the information about the grid client.
- UVOS Gridmap Policy Decision Point allows administrator to store certificate to local account mappings in the UVOS server. This can effectively replace the traditional gridmap file. The mappings are stored as scoped attributes of a user, and a group (where the attribute is valid) is used to distinguish the mappings for different grid nodes if needed.
- UVOS Access Policy Decision Point which provides a possibility to perform a fine grained authorization decisions based on the attributes received from the UVOS server.

The Policy Information Point (PIP) and Policy Decision Point (PDP) terms come from the security domain and have a precise meaning in the Globus Toolkit

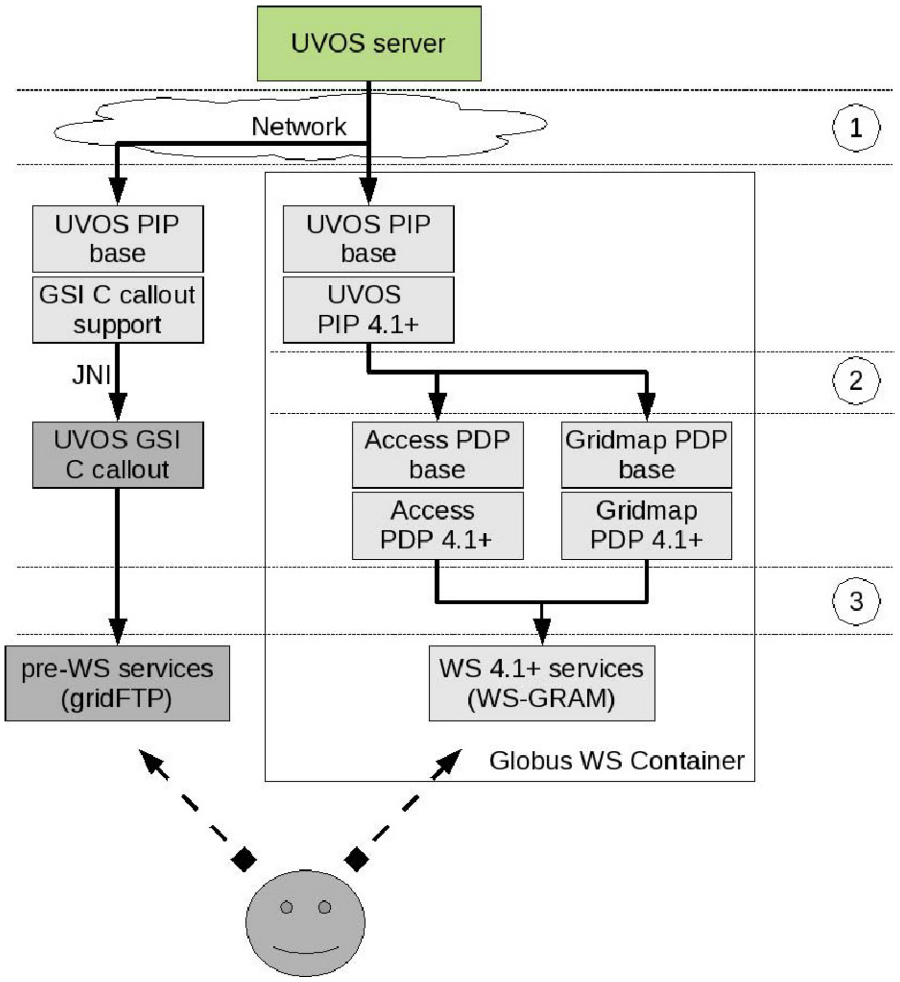


Fig. 2. The figure shows the architecture of the UVOS authorization subsystem for the version 4.1+ of the Globus Toolkit. All components which are specific to that particular Globus version are marked. Solid arrows shows information flow through the system. Dashed lines shows service invocations as performed by a user. Light boxes are Java components and dark gray boxes are C modules. In the figure there are three layers of communication marked. In the layer (1) communication is done through a network. The Policy Information Point (PIP) queries the UVOS server for the user's attributes. Communication in the next layer (2) is performed internally by the Globus security stack which passes assembled information about the client to the authorization modules. Eventually in the layer no (3) an authorization decision and (optionally) a mapping to a local account is passed again by the security stack to the invoked service (or more formally to the Globus policy enforcement code which protects the service access).

authorization API. The PIP is responsible for assembling grid client's credentials (as for example its identity, attributes) and the PDP makes an authorization decision (using the data provided by the PIPs). In fact this design is used only for Globus web services. The authorization API for the legacy services is much simpler — it allows only to create a callout which makes an authorization decision.

All three elements of our project are available for the Globus web services only. In the case of the legacy services only the UVOS Gridmap PDP element is available. It encapsulates the PIP functionality. The more fine grained access control would be clearly service specific so we do not plan to develop more features for our pre-WS Globus authorization module.

The security infrastructure of the Globus WS container allows for using multiple PIPs and PDPs together. The configuration is flexible with the changes introduced with the Globus Toolkit 4.1. Among others it is possible to choose an algorithm which makes a final access/deny decision taking as an input the decisions of the individual PDPs. As our modules may be installed individually we can achieve a high flexibility, e.g. by mixing UVOS and classic gridmap file authorization.

The implementation of our software consists of multiple small modules. The base implementation of the UVOS PIP was done using the Java language so it was possible to make use of the UVOS client library. As this implementation must work in different environments, there are three wrappers for the Globus Toolkit 4.0, 4.1+ and for the C security API. In the last case the Java Native Interface technology was used to connect both technologies. The PDP modules were designed in an analogous way, with the single exception of the code for the pre-WS services. In this case there is only one module which incorporates a C PIP wrapper. The whole architecture of our prototype is presented in the fig. 2.

6 Summary

We have designed and developed a prototype of a complete authorization solution for Globus Toolkit 4.x interoperable with the UNICORE 6. It uses the UVOS server as a backend. The pull style of attributes acquisition is used. One of the important results of our work is support for the GridFTP authorization which is absent in the GridShib system. Additionally, a native and deep integration with the Globus security stack allows for a more flexible usage of our modules with the other ones.

The future work will be carried out in multiple directions. In order to produce a production ready implementation we will have to provide solutions for reliability, that is solutions for the proper operation of the system during the failure of a communication with the UVOS server. Later plan to verify the interoperability of the system (and also the whole UVOS system) with the Shibboleth 2 middleware. Further integration of our solutions with the GridShib is planned but will be possible after a SAML 2.0 compatible version of GridShib will be released. Eventually providing a similar solutions for the other leading grid middlewares, gLite and NorduGrid ARC can be seen as an ultimate goal.

Acknowledgements. This work is supported by European Commission under IST grant Chemomentum (No. 033437). The financial support of the eea grant Kardionet (No. PL-0262) is also ackowledged.

References

1. Foster, I.T., Kesselman, C., Tuecke, S.: The Anatomy of the Grid — Enabling Scalable Virtual Organizations. In: Computing Research Repository (CoRR), vol. cs.AR/0103025 (2001), http://arxiv.org/abs/cs.AR/0103025 (2009)
2. Alfieri, R., et al.: From gridmapfile to voms: managing authorization in a grid environment. Future Generation Comp. Syst. 21(4) (2005)
3. The Shibboleth system webpage (May 2009), http://shibboleth.internet2.edu
4. The UVOS project (May 2009), http://uvos.chemomentum.org
5. Cantor, S., et al. (eds.): Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard (March 15, 2005), http://docs.oasis-open.org/security/saml/v2.0/ (May 2009)
6. Barton, T., et al.: Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, Gridshib, and MyProxy. In: 5th Annual PKI R&D Workshop (April 2006), http://grid.ncsa.uiuc.edu/papers/gridshib-pki06-final.pdf (May 2009)
7. Foster, I., et al.: The Open Grid Services Architecture, Version 1.5. GGF 2006 (2006), http://forge.gridforum.org/projects/ogsa-wg (May 2009)
8. The Virtual Data Toolkit project (May 2009), http://vdt.cs.wisc.edu/index.html
9. The Grid User Management System (May 2009), https://www.racf.bnl.gov/Facility/GUMS/1.3/index.html
10. Groeper, R., et al.: A concept for attribute-based authorization on D-Grid resources. Future Generation Comp. Syst. 24(3) (2009)
11. The Chemomentum project (May 2009), http://www.chemomentum.org
12. Hughes, J., et al. (eds.): Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard (March 15, 2005), http://docs.oasis-open.org/security/saml/v2.0/ (2009)
13. Scavo, T. (ed.): SAML V2.0 Deployment Profiles for X.509 Subjects, OASIS Committee Specification (March 27, 2008), http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml2-profiles-deploy-x509-cs-01.pdf (May 2009)
14. Venturi, V., Scavo, T., Chadwick, D.: OGSA Attribute Exchange Profile Version 1.2. Open Grid Forum (2007)

Using UNICORE and WS-BPEL for Scientific Workflow Execution in Grid Environments

Guido Scherp[1], André Höing[2], Stefan Gudenkauf[1], Wilhem Hasselbring[1], and Odej Kao[2]

[1] OFFIS Institute for Information Technology, R&D-Division Energy, Escherweg 2, 26121 Oldenburg, Germany
{stefan.gudenkauf,guido.scherp}@offis.de, wha@informatik.uni-kiel.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. Within the BIS-Grid project[1], a BMBF-funded project in the context of the German D-Grid initiative, we developed the BIS-Grid workflow engine that is based upon service extensions to UNICORE 6 to use an arbitrary WS-BPEL workflow engine and standard WS-BPEL to orchestrate stateful, WSRF-based Grid services. Although aimed at proving the feasibility of applying Grid technologies for business information systems integration, we illustrate that this engine is also well-suited for scientific workflow execution, making standard WS-BPEL-based tooling accessible for scientific workflows.

In this paper, we describe using the BIS-Grid engine for the execution of scientific workflows. This includes a differentiation of scientific and business workflows in general and an analysis of the suitability of the BIS-Grid infrastructure to execute scientific workflows. We propose reusable WS-BPEL patterns for typical scientific workflow activities whereas job submission is focused. Finally, we prospect our future work.

1 Motivation

Modern Grid middlewares such as UNICORE 6[2] are based on the Web Service Resource Framework (WSRF)[3], a standard that extends classical, stateless Web services to be stateful. Like Web services, WSRF-based Web services, also called Grid services, can be orchestrated to form complex workflows that itself are provided as services by utilizing the Web Service Business Process Execution Language (WS-BPEL). Although originally developed for service orchestration in the business domain, WS-BPEL gained much attention from scientific communities to be adopted for the design and execution of scientific workflows.

[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.

[2] `http://www.unicore.eu`

[3] `http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf`

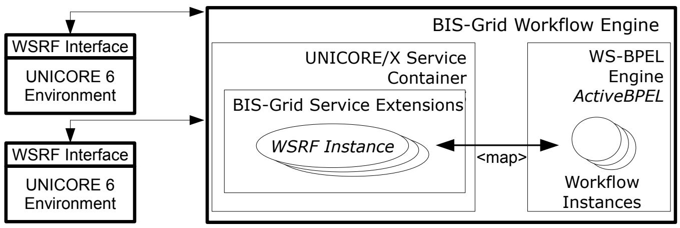


Fig. 1. Overview of architecture of the BIS-Grid workflow engine

Within the BIS-Grid[4] project, a BMBF-funded project in the context of the German D-Grid initiative, we developed the BIS-Grid workflow engine that is based upon service extensions to UNICORE 6 to use an arbitrary WS-BPEL workflow engine and standard WS-BPEL[5] to orchestrate Grid services. These service extensions act as a WSRF proxy to the functionalities of the original WS-BPEL engine and to the deployed WS-BPEL workflows itself, cp. Fig. 1, providing a Workflow Management Service for workflow deployment and a generic Workflow Service for workflow execution and monitoring. For a more in-depth view on the architecture of this engine, see [10]. Although originally aimed at proving the feasibility of applying Grid technologies for the integration of business information systems, this engine is also suited for scientific workflow execution, making standard WS-BPEL-based tooling accessible for scientific workflow execution. To hide workflow complexity from the scientific user, we propose reusable WS-BEL patterns for typical scientific workflow activities such as job submission and data transfers instead of specific language extensions.

The paper is organized as follows. Related work is discussed in Sec. 2, followed by a short overview of the differences between scientific and business workflows in Sec. 3. Section 4 discusses the principal requirements for scientific workflow execution and how they are addressed by BIS-Grid. The use of WS-BPEL for scientific workflows including our WS-BPEL pattern is shown in Sec. 5 by the example of job submission. Section 6 provides an outlook on our future work, and Sec. 7 provides a conclusion.

2 Related Work

The Chemomentum project already provides workflow extensions for UNICORE 6, consisting of two UNICORE 6 service containers. The first represents a workflow engine that processes workflows on a logical level, the second represents a service orchestrator that transforms so-called Work Assignments into jobs, given in the Job Submission Description Language (JSDL) [1]. Both, this UNICORE 6 workflow system and the BIS-Grid engine, are implemented

[4] `http://www.bisgrid.de`

[5] I.e., we did not modify nor extend the WS-BPEL language.

as service extensions to the UNICORE 6 service container. However, the UNICORE 6 workflow system does not support the integration of a WS-BPEL workflow engine.

Akram et al. [2] identify requirements for scientific workflows – namely modularity, exception handling, mechanisms compensation/recovery, adaptivity and flexibility, and workflow management – by the example of a protein crystallography workflow. They also describe how the BPEL language addresses these requirements, and the shortcomings of BPEL for scientific workflows. Most prominently, these are the limited adaptivity regarding workflow modifications at run-time, the lack of support for user interactions by the BPEL specification, and the need to wrap non-portable engine-specific workflow management capabilities using appropriate standards in order to use them in a portable manner.

Regarding the use of BPEL for Grid service orchestration, Leymann proposes BPEL4WS[6] as foundation since it already fulfills many requirements of the WSRF standard [12]. The appropriateness of BPEL is also examined and confirmed in [5], [6], [7], [8], and [14]. These works mainly focus on scientific workflows and, except for Ezenwoye et al. [8], rely on extending or adapting BPEL, thus creating dialects.

The execution of jobs with WS-BPEL is also discussed in [15] in which a two-stage approach is proposed. In the first stage a base flow is modeled to define job execution, supplemented by a JSDL job description and a fault-handling policy based on WS-Policy[7]. This base flow is expanded automatically in the second stage by additional WS-BPEL fault-handling activities corresponding to the respective fault-handling policy. The execution of the workflow is based on two further non-WS-BPEL services, a job proxy to encapsulate job execution and to receive notification messages from a scheduling system, and a fault-handling service to apply extended fault-handling strategies such as workflow instance migration. The approach was implemented and tested on IBM software. In [16], Zhao et al. present a visual tool that abstracts a typical sequence of BPEL activities for scientific computing to a new single activity. This sequence comprises steps like *submitTask* or *getTaskStatus* and looks slightly similar to the job submission workflow we present in Sec. 5. However, Zhao et al. focus on visual complexity in the workflow editor. Before workflow deployment, the proprietary code is translated to standard WS-BPEL.

3 Scientific vs. Business Workflows

A comparison between scientific workflows and business workflows is the topic of several publications – directly or indirectly, as, for example, in [2] and [3]. Thus, we will not present a complete comparison but focus on the principal differences of scientific and business workflows, see Tab. 1.

[6] BPEL4WS 1.1 is the predecessor of WS-BPEL 2.0.

[7] `http://www.w3.org/Submission/WS-Policy/`

Table 1. Scientific vs. business workflows

Scientific workflows	***Business workflows***
Data-driven Control flow is implicit; an activity starts when the required input data is available.	*Control-driven* Data flow is implicit and data is manipulated when the corresponding activity is executed in the control flow.
User-centric Rights are often associated to persons (scientists) directly. The workflow designer is often also the workflow executor (does not necessarily hold for e-Science).	*Role-centric* Rights are associated to roles that are associated to persons. Elaborate role models for workflow participants/stakeholders.
Voluminous data handling Data handling often requires third-party transfers (data transfers between two remote servers that are initialized by a local client).	*Information handling* Workflow data is usually small, regarded as *information* and is stored in process variables during workflow execution.
Experiment implementation Monitoring is of great importance, especially for intermediary results of a workflow. Workflows tend to evolve quickly as knowledge on the domain/workflow is collected (cp. [3]).	*Service provisioning* Workflows must be guaranteed to complete and provide results as advertised to and contractually agreed with customers [3].

4 BIS-Grid Engine for Scientific Workflows

As depicted in Sec. 2, WS-BPEL becomes more and more important for the scientific community regarding the execution of scientific workflows in Grid environments. Modern Grid middlewares such as UNICORE 6 and Globus Toolkit 4 provide their functionalities – for example, data transfer and job submission – as (WSRF-based) Grid services. Scientific workflows that build on these Grid middlewares must be described as an ordered invocation of such Grid services. WS-BPEL, as the de facto standard for Web service orchestration, comes naturally in mind for Grid service orchestration, although originally being designed for business workflows.

Per se, WS-BPEL has some shortcomings that constrains its usability for scientific workflows. Originally, the language has been designed to orchestrate stateless Web services. Since data transfer and job execution, for example, have state, the WSRF standard was developed to enable stateful Web services (Grid services). Consequently, Grid service invocations are much more complex than standard Web services – a WSRF service instance has to be created, is used, and finally must be destroyed. To address this, we developed appropriate WS-BPEL patterns for Grid service invocations for the Grid middlewares UNICORE 6 and Globus Toolkit 4 [4,11]. Further shortcomings do not originate from the WS-BPEL language directly but from the workflow execution environments and workflow design tools. Available WS-BPEL workflow engines, open source or commercial, are not fully interoperable with the security features of existing Grid middlewares. Such features are, for example, the support of SAML assertions [13] to present additional signed security tokens (as roles), or the support of the Grid Security Infrastructure (GSI) of Globus Toolkit 4 infrastructures that rely on proxy certificates. Furthermore, available WS-BPEL design tools are usually not suitable for scientific users because the applied workflow model is close to the technical WS-BPEL language. Considering this, scientific users require a workflow model fitting to their domain. For example, scientific users may need to run several computations on selected data in a specific order by dropping boxes (computations) onto a workbench and drawing lines (data flow) between them. Prospects on these issues are presented in Sec. 6.

Table 2 presents an overview of the principal requirements we identified for scientific workflows, and presents which of them are addressed by the WS-BPEL language and tooling, or by the BIS-Grid engine. As shown, RQ-1 is met by WS-BPEL itself. Regarding monitoring (RQ-2), the BIS-Grid engine supports mechanisms that base upon the propagation of the monitoring capabilities of the internal WS-BPEL engine by the UNICORE 6 layer, see Fig. 1, [10], and [11]. Upon these, advanced capabilities such as pull- or push/notification-based monitoring of workflow activities can be implemented. Design-time error handling and compensation (RQ-3 and RQ-4) are met by WS-BPEL, while run-time error handling and compensation are matters of the workflow execution environment. While these are important issues to be addressed in operational execution environments, they are not focused in BIS-Grid and regarded as underlying the actual workflow execution engine. Regarding RQ-5, BIS-Grid relies on the the Netbeans IDE[8] and it's BPMN-oriented visual DSL (Domain-specific language). Together with appropriate WS-BPEL patterns, this represents an abstraction from the technical workflow implementation that is comfortable for general purpose workflow design both for business workflows as well as scientific workflows, and provides a basis for further domain-specific abstraction above the WS-BPEL pattern layer. RQ-6, voluminous data transfers, is addressed in the following sections in terms of *file staging.*

5 WS-BPEL Job Submission Pattern for Scientific Workflows

Scientific workflows often are realized as (batch) jobs that can be defined as non-interactive computational tasks that are intended to be executed on high-performance computing (HPC) systems. Grid middlewares typically support the submission of such jobs to an HPC system by utilizing the local batch system, but modern Grid middlewares also provide means for exposing their functionalities as Grid services, thus enabling service orchestration. Grid services such as job submission services support further standards like the Job Submission Description Language (JSDL) or OGSA Basic Execution Services (OGSA-BES)[9]. Using job submission services results in almost generic sequences of Grid and/or Web service invocations[10] that can be encapsulated in a generic and configurable WS-BPEL (sub-)workflow. Our BIS-Grid engine allows to execute such a workflow in Grid environments whereas the workflow itself is provided as a Grid service. This facilitates reuse in higher-level service orchestrations. In [4] we identified WS-BPEL patterns for orchestrating Grid services using standard WS-BPEL.

[8] `http://www.netbeans.org/features/soa/index.html`

[9] `http://www.ogf.org/documents/GFD.108.pdf`

[10] For most jobs, these sequences are almost identical, cp. [16].

Table 2. Principal requirements of scientific workflows

RQ	*Requirement description*	*Compliance*
RQ-1	*Modularity and composability.* Although evolving in nature, scientific workflows often base upon recurring standard activities. Regarding different levels of abstraction, providing sub-workflows as standard activities of superior workflows facilitates reuse and maintenance. Separating workflows in different parts facilitates the scalability of workflow execution.	WS-BPEL is composable by nature.
RQ-2	*Monitoring.* Scientific workflows are often long-running; progress monitoring and the inspection of intermediary results is therefore an important issue.	The BIS-Grid engine supports basic monitoring of workflow state. WS-BPEL `EventHandlers` improve simple execution state monitoring.
RQ-3	*Error handling and fault tolerance.* The long-running nature of scientific workflows causes interruption due to errors to be regarded as highly undesirable. Mechanisms should ideally address design-time (error handling for forseen events) and run-time (fault tolerance).	WS-BPEL provides mechanisms for error handling at design time.
RQ-4	*Adaptability.* As the underlying resource infrastructure may change during workflow execution, workflow adaptability is regarded as desirable. Mechanisms should ideally address design-time (compensation for forseen events) and run-time.	WS-BPEL provides mechanisms for compensation at design-time.
RQ-5	*Domain-specificity.* Often, scientists are not only the users of scientific workflows but also their designers. This requires adequate domain-specific modeling while technical details should be concealed as far as possible.	BIS-Grid uses Netbeans IDE for workflow design, using it's BPMN-like visual DSL and appropriate WS-BPEL patterns [4,11] to abstract from technical workflow implementation.
RQ-6	*Voluminous data transfers.* Scientific computations are often based on voluminous data. This data has to be transferred to the respective computing resources.	Generally, third party transfers can be modeled and executed with BIS-Grid. JSDL-compliant Grid middlewares provide file staging mechanisms as defined by JSDL.

Table 3. Job Submission Pattern

	Pattern description
Motivation	Scientific workflows often are designed as non-interactive computational tasks in the form of (batch) jobs. Regarding reuse, there is the need to integrate such jobs in workflows that are designed on a higher level of abstraction than jobs.
Intention	Define a workflow that executes a (batch) job on a UNICORE 6 installation by using the target system service and it's job management service to submit and start a job and its respective data, and to retrieve the job's outcome upon completion.
Behavior	See Figure 2.
Participants	The invoker of the job submission workflow, the Target System Factory Service, the Target System Service, and the Job Submission Service.
Consequences	(1) Job submission is encapsulated in a workflow using an appropriate workflow description language. This facilitates the reuse of existing jobs on a higher level of abstraction and reduces submission errors and redundant user-triggered submissions by reuse. (2) Workflow reuse also simplifies the protocol of high-level workflows and abstracts from submission details. When the workflow language allows hierarchical composition, as WS-BPEL, the job submission may be described with the same language as the high-level workflows.

Based on this pattern we developed a WS-BPEL pattern to encapsulate Grid service invocations for job submission to UNICORE 6. A description of this job submission pattern is presented in Tab. 3.

Job submission to UNICORE 6 consists of several phases that are described briefly in the following. Additionally, Fig. 2 illustrates the corresponding workflow. Please note that we omitted exception handling and compensation for the

Table 4. Signature of the job submission workflow

Name	*Type*	*Description*
	Input parameters	
TargetSystem (mandatory)	wsa:EndpointReferenceType	The endpoint to the Target System Factory Service that should be used to create the Target System Service instance.
JSDL (mandatory)	jsdl:JobDefinition_Type	The job description according to the JSDL standard.
LifetimeIntervall (optional)	xsd:duration	Maximum runtime of the job execution. If omitted, a default lifetime is used.
WaitInterval (optional)	xsd:duration	Waiting time between two job execution status retrievals. If omitted, a default waiting time is used.
	Output parameters	
Result	xsd:string	The result (successful or failed) of the job execution.

sake of clarity. The current signature of the job submission workflow is described in Tab. 4. At least, the endpoint to a UNICORE 6 target system and the JDSL Job description is mandatory as input. At the moment, the output is a job failed/succeeded message. This workflow itself can be used in high level workflows in which a resource broker is invoked to choose a target system with least load before submitting a job.

1. `Job Submission Receive`: A JSDL job description and configuration parameters are received and stored in process variables.
2. `Target System Service Instance Create`: A Target System Service instance is created via the default factory instance of the Target System Factory Service.
3. `Target System Service Instance Submit Job`: The JSDL job description is submitted to the Target System Service instance which creates a Job Management Service instance.
4. `Job Management Service Instance Start Job: The Job Management` Service instance is used to start job execution.
5. `Job Management Service Instance Retrieve Result`: The status of the job execution is fetched periodically from the Job Management Service until the job is completed or failed, afterwards the job result is stored in a process variable.
6. `Job Management Service Instance Destroy`: The Job Management Service instance is destroyed.
7. `Target System Service Instance Destroy`: The Target System Service instance is destroyed.
8. `Job Result Reply`: The job result is returned.

The JSDL standard includes the description of file staging (RQ-6) to initiate file transfers before (stage-in) and after (stage-out) the actual job execution. This mechanism can be used to transfer input and output data (often several

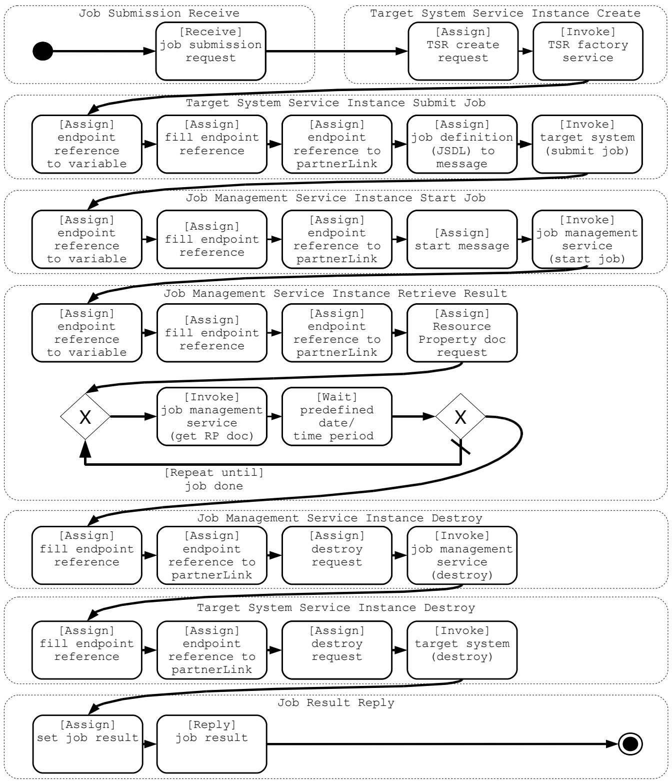

Job Submission Receive
[Receive] job submission request
Target System Service Instance Create
[Assign] TSR create request
[Invoke] TSR factory service
Target System Service Instance Submit Job
[Assign] endpoint reference to variable
[Assign] fill endpoint reference
[Assign] endpoint reference to partnerLink
[Assign] job definition (JSDL) to message
[Invoke] target system (submit job)
Job Management Service Instance Start Job
[Assign] endpoint reference to variable
[Assign] fill endpoint reference
[Assign] endpoint reference to partnerLink
[Assign] start message
[Invoke] job management service (start job)
Job Management Service Instance Retrieve Result
[Assign] endpoint reference to variable
[Assign] fill endpoint reference
[Assign] endpoint reference to partnerLink
[Assign] Resource Property doc request
X
[Invoke] job management service (get RP doc)
[Wait] predefined date/ time period
X
[Repeat until] job done
Job Management Service Instance Destroy
[Assign] fill endpoint reference
[Assign] endpoint reference to partnerLink
[Assign] destroy request
[Invoke] job management service (destroy)
Target System Service Instance Destroy
[Assign] fill endpoint reference
[Assign] endpoint reference to partnerLink
[Assign] destroy request
[Invoke] target system (destroy)
Job Result Reply
[Assign] set job result
[Reply] job result

Fig. 2. Job submission process

gigabytes) for a single job execution. Hence, the corresponding job submission workflow does not include explicit file transfer activities. Note that the proposed pattern is not directly applicable to all kinds of scenarios. For example, a scientific user may want to run several jobs using the same input data that should not be transfered for each single job execution. A solution to this scenario is to provide a special file transfer workflow that can be directly integrated in the job submission workflow, or in higher-level workflows (cp. Sec. 6).

6 Prospects

As already stated in Sec. 5, file staging for compute jobs is implicitly possible (cp. RQ-6). The staging is defined in JSDL, placing the responsibility for pre- and post-job execution of data transfers on the execution environment. More complex scenarios, however, require more sophisticated functionalities. For example, this is the case for explicit data staging as input for a bundle of jobs. Generally, file staging requires to transfer files from a source to a destination – supported by appropriate protocols such as GridFTP or UNICORE File Transfer Services.

Typical file transfers are executed between a local user client and a remote destination, or between a remote source and a remote destination, the latter being referred to as *third-party transfers*. Since available workflow engines usually are not designed to pass and transfer voluminous data (RQ-6) directly, third-party transfers can be realized by workflows – using the workflow engine solely for transfer coordination. We propose to regard such transfer workflows as a single, configurable workflow activity specific to the scientific domain (cp. RQ-5).

We developed a WS-BPEL pattern to invoke Grid services in Globus Toolkit 4 (GT4) [4]. This pattern provides a basis to develop GT4 job submission and file transfer workflows analogous to those discussed in this paper. However, beside mere service orchestration it is necessary to address the respective security infrastructures. Since the BIS-Grid engine is based on UNICORE 6, the secure invocation of (external) UNICORE 6 Grid services in workflows is guaranteed. To enable the secure invocation of GT4 Grid services we plan to support the GT4 Grid Security Infrastructure (GSI) in the BIS-Grid engine. This will be evaluated in an appropriate application scenario which is currently being prepared. Another important issue for scientific workflows is scalability, which is already considered in the design of the BIS-Grid engine [9]. Nevertheless, scalability is currently not supported directly and thus regarded as future work.

7 Conclusion

In this paper, we described to use the BIS-Grid workflow engine, consisting of service extensions to UNICORE 6 and an arbitrary WS-BPEL engine, for the execution of scientific workflows. Thereby we focused on job submission as an important aspect of scientific workflows, and presented an appropriate WS-BPEL pattern for job submission with UNICORE 6. Previously, we discussed the principal differences of scientific and business workflows, and presented the principal requirements of scientific workflows. We also presented our future work focusing on advanced file staging mechanisms, on interoperability with Globus Toolkit 4 by supporting the Grid Security Infrastructure and by developing analogous WS-BPEL patterns specific to Globus Toolkit 4, and on regarding scalability.

References

1. Job Submission Description Language (JSDL) Specification, Version 1.0 (November 2005), `http://www.gridforum.org/documents/GFD.56.pdf`
2. Akram, A., Meredith, D., Allan, R.: Evaluation of BPEL to Scientific Workflows. In: CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, pp. 269–274. IEEE Computer Society, Los Alamitos (2006)
3. Barga, R., Gannon, D.: Scientific versus Business Workflows. In: Workflows for e-Science, pp. 9–16. Springer, London (2007)
4. Brinkmann, A., Gudenkauf, S., Hasselbring, W., Höing, A., Kao, O., Karl, H., Nitsche, H., Scherp, G.: Employing WS-BPEL Design Patterns for Grid Service Orchestration using a Standard WS-BPEL Engine and a Grid Middleware. In: Bubak, M., Turala, M., Kazimierz, W. (eds.) CGW'08 Proceedings, Cracow, Poland, pp. 103–110. ACC CYFRONET AGH (2009)

5. Chao, K.-M., 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'04), vol. 01, p. 284. IEEE Computer Society, Los Alamitos (2004)
6. Dörnemann, T., Friese, T., Herdt, S., Juhnke, E., Freisleben, B.: Grid Workflow Modelling Using Grid-Specific BPEL Extensions (2007)
7. 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)
8. Ezenwoye, O., Masoud Sadjadi, S., Cary, A., Robinson, M.: Orchestrating WSRF-based Grid Services. Technical report, School of Computing and Information Sciences, Florida International University (April 2007)
9. Gudenkauf, S., Hasselbring, W., Heine, F., Höing, A., Scherp, G., Kao, O.: BisGrid: Business Workflows for the Grid. In: CGW'07 Proceedings, Krakow, Poland, pp. 86–94. ACC CYFRONET AGH (2008)
10. Gudenkauf, S., Hasselbring, W., Höing, A., Scherp, G., Kao, O.: Workflow Service Extensions for UNICORE 6 - Utilising a Standard WS-BPEL Engine for Grid Service Orchestration. In: César, E., et al. (eds.) Euro-Par 2008. LNCS, vol. 5415, pp. 103–112. Springer, Heidelberg (2009)
11. Gudenkauf, S., Höing, A., Scherp, G.: Catalogue of WS-BPEL Design Patterns. Technical report (May 2008)
12. Leymann, F.: Choreography for the Grid: towards fitting BPEL to the resource framework: Research Articles. Concurr. Comput.: Pract. Exper. 18(10), 1201–1217 (2006)
13. Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., Scavo, T.: Security Assertion Markup Language (SAML) V2.0 Technical Overview. Working Draft (February 2007), `http://www.oasis-open.org/committees/download.php/22553/sstc-saml-tech-overview-2`
14. Slomiski, A.: On using BPEL extensibility to implement OGSI and WSRF Grid workflows: Research Articles. Concurr. Comput.: Pract. Exper. 18(10), 1229–1241 (2006)
15. Tan, W., Fong, L., Bobroff, N.: BPEL4Job: A Fault-Handling Design for Job Flow Management. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 27–42. Springer, Heidelberg (2007)
16. Zhao, Z., Zhang, R., Lin, J., Chen, Y., Zhang, H., Li, L.: An Improved Visual BPEL-Based Environment for Scientific Workflow. In: GCC '08: Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing, Washington, DC, USA, pp. 435–441. IEEE Computer Society, Los Alamitos (2008)

Enhancing UNICORE Storage Management Using Hadoop Distributed File System

Wasim Bari[1], Ahmed Shiraz Memon[2], and Bernd Schuller[2]

[1] Rheinisch-Westfälische Technische Hochschule (RWTH)
52056 Aachen, Germany
[2] Institute of Advanced Simulation, Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
D-52425 Jülich, Germany

Abstract. UNICORE is a state of the art and well tested Grid middleware, designed for seamless and secure access to distributed resources, applications and data, in an easy to use fashion. A wide variety of UNICORE applications for example in bio-informatics generate and compute huge amounts of data. These large amounts of data are not easy to manage reliably and efficiently with the default UNICORE storage system which is using a standard file system. Hence, the current UNICORE does not support a scalable distributed storage system so far. We have integrated Apache Hadoop and its supported distributed storage/file systems into the UNICORE storage management service. Thus allows to build a UNICORE storage system providing data replication, disaster recovery, durability and elasticity. In this paper we will present the architecture and operation of a prototype called UniHadoop, which provides the integration of UNICORE and the distributed storage systems (DSS) supported by Hadoop and highlight its potential in usage scenarios.

1 Introduction

Computational power has increased significantly, typical desktop systems are now far superior to the super computers in the last years, and the current high-performance systems have reached the PetaFlop/s scale. This allowed to tackle new computational challenges in applications such as high-energy physics, nuclear physics, weather forecasting, data mining, environmental modeling, which can now be executed in 'feasible' time spans. These applications usually produce data with rapid growth and in large volumes. For example, the High Energy and Nuclear Physics (HENP) data volume will rise from hundreds of petabytes to exabytes (10^{18}) from 2012-2015 [1]. These huge chunks of data, produced from such applications, need to be stored, exchanged, visualized and analyzed for a variety of purposes. This requires specialized storage systems with high data availability, reliability and distribution.

UNICORE[1] is used as underlying Grid middleware by a number of scientific applications [2]. At the time of writing, its storage management interfaces do not

[1] **Uni**form **I**nterface to **Co**mputing **Re**sources.

support distributed storage system interaction [3], but are limited to file system access. In this paper we present the design and integration of Apache Hadoop and its supported storage systems with UNICORE. The remainder of this paper is structured as follows. In Section 2 UNICORE's architecture is elaborated in detail. Section 3 gives an overview of Hadoop, its design and supported storage system. in Section 4 we present UniHadoop, i. e. the integration of UNICORE and Hadoop supported DSS. After discussing the usage scenarios, the paper ends with a brief conclusion and related work.

2 UNICORE Architecture

UNICORE is a ready-to-run, Open Source Grid middleware that enables seamless, secure and transparent access to resources. It hides unnecessary details from the users and provides single sign-on with well-established security mechanism. UNICORE offers a number of clients for job creation, submission and monitoring. In its most recent version UNICORE 6 conforms to the Open Grid Service Architecture (OGSA) [4] and several open Grid standards as mandated by the Open Grid Forum (OGF) [5]. It is a Web services based implementation using the Web Services Resource Framework (WSRF) [6]. The user can use various clients for jobs creation, submission and monitoring, both graphical and command-line oriented.

From the conceptual point of view, one can divide UNICORE in three tiers: Client, Server and Target. The user logs into the client and creates the job. The job along with user information is authenticated by the Gateway and forwarded to the Web services interfaces offered by the UNICORE/X server. These services interact with XNJS[2] which interacts with XUUDB[3] for authorization information. Finally, task is handed over to either Target System or Target System Interface (TSI) which then executes the job.

A Client layer: With growing use of UNICORE and emergence of new applications and problem domains, a set of clients is supported [7]. It helps in job definition and application integration from different domains in an easy to do manner. These are command line as well as GUIs. Java is used as development language, so it supports many platforms.UNICORE Commandline Client(UCC), Grid Programming-based Environment(GPE) Client, UNICORE Rich Client, Web Portal Client and High Level API (HILA) clients be used to communicate with UNICORE server and underlying resources.

Service layer: The core of UNICORE consists of services and components compliant with the OGSA model and is based on WSRF 1.2, SOAP [8] and WS-I standards [9]. Analyzing it from top to bottom, request first encounters with Gateway which acts as a secure entry point for any number of UNICORE sites. Client requests are authenticated and encrypted on this level using X.509 certificates [10]. Next the central job controlling entity of UNICORE is XNJS which provides a

[2] e**X**tended **N**etwork **J**ob **S**upervisor.
[3] e**X**tended **U**NICORE **U**ser **D**ata **B**ase.

set of services like job management, storage management and data transfer services. It communicates with the XUUDB for authorization of users, maps the jobs to the desired Target Systems using Incarnation Database (IDB),submits jobs and monitors its progress. It does not only support OGSA-* based open standards but also has its own proprietary interfaces named UNICORE Atomic Services (UAS). Storage Management Service (SMS) and File Transfer Service (FTS), part of UAS, exposes storage resources to users and provides different operations ranging from storage management to data transfer. Target System Registry Service (TSR) is another essential component like in any SOA system. Any client wishing to utilize UNICORE 6 must have information regarding available services and their description in a specific Grid. The detailed information is published in TSR. It works as a single point of entry for clients. A TSR is shareable between sites.

System layer: The Target System Interface (TSI) lies at bottom in UNICORE architecture. Its role is to communicate between the local operation system and XNJS. The TSI takes the concrete jobs from the XNJS and executes them on the target system as the local user determined by the XNJS. A temporary working directory named USpace is attached with each job. Every job has its own USpace which stores data related to the specific job and later user can transfer data to and from storage system (or the client) to the USpace.

3 Hadoop

The Web search engine giant Google processes and stores huge data and thus requires a storage system with special attributes for instant replies. In 2003, Google described its highly scalable, fault tolerant, dynamic distributed file system called the Google File System (GFS) [11]. This file system is designed to run on commodity machines and can respond to thousands of queries per second. Component failure was taken as the norm rather than the exception. Files are getting huge as compared to traditional files, increased computation power consistently producing more and more data which needs to be organized for future analysis. It was observed that many applications follow the “Write Once Read Many” paradigm. The GFS is proprietary software.

The Hadoop distributed file system (HDFS) is an open source implementation of the ideas and algorithms of Google’s GFS, realised in Java [12]. It is used by many organizations, notably Yahoo, where it has been proven to scale up to 4000 nodes and 16PB disk size.

Apart from the HDFS, Hadoop offers an implementation of the MapReduce data processing framework, again as published in a seminal paper by Google [13].

3.1 HDFS Architecture and Supported File/Storage Systems

The Hadoop distributed file system follows a Master/Slave architecture. One node in the HDFS cluster works as NameNode, while the others serve as DataNodes. The NameNode manages all the meta data regarding filesystem like file namespace, file to block mapping, block location information, access control version, block version numbers etc. As shown in Fig. 1, NameNode runs on a separate machine and periodically communicates with the DataNodes but this communication is least started by NameNode. It is the DataNode's responsibility to send regular heart beat messages to the NameNode.

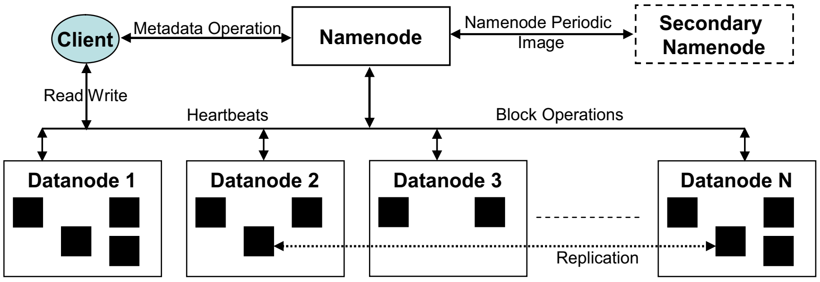


Fig. 1. HDFS Architecture showing node interdependencies

The NameNode starts up in "safe mode", no changes are possible in this mode and it remains in it as per configuration. As the NameNode starts up, each DataNode sends a heartbeat message to the NameNode. After that the NameNode asks each DataNode to send its block report. The DataNode computes the block report and returns it to the NameNode. After this the NameNode exits the safe mode and goes into "normal mode" if all conditions are fulfilled.

During the normal mode and with periodic communications NameNode gets certain pieces of information such as: if some DataNode is down, any disk failure on a DataNode, checks the Replica status of files, which DataNode stores which replica. Finally, Hadoop also supports a number of open source as well as commercial file/storage systems other than HDFS.

4 UniHadoop

UniHadoop is the integration of multiple Hadoop supported distributed storage systems with UNICORE 6. As discussed in section 2, UNICORE is a service oriented, flexible system with a number of proprietary and open standard services. The SMS manages the storage resources and initiates the data transfer with the FTS. In the UniHadoop prototype, SMS and FTS were extended to allow the integration of Hadoop with UNICORE. After the integration, user can access any of the Hadoop supported storage system without altering call interface to UNICORE.

4.1 UniHadoop Architecture

As UNICORE itself, UniHadoop is realized in Java and uses WS-RF based Web services. Fig. 2 shows the architecture of the UniHadoop prototype. An abstract

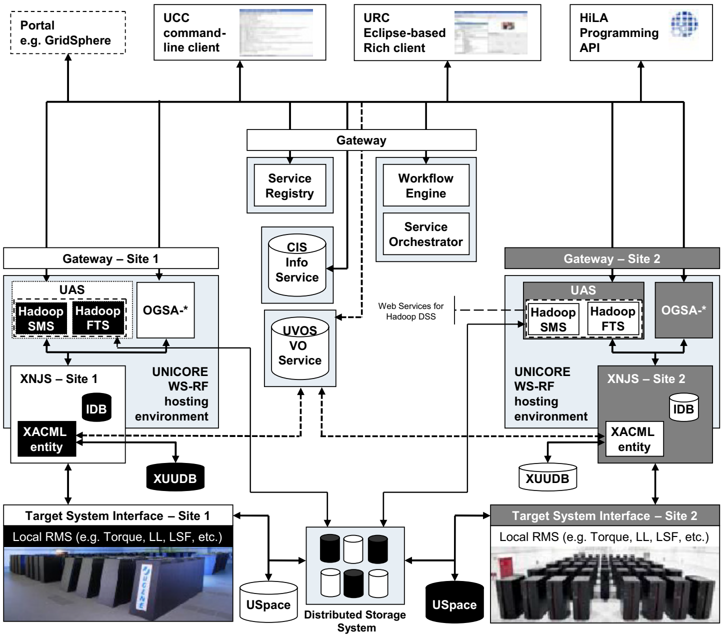


Fig. 2. UniHadoop Architecture

file system object is used for genericity. New Web services, HadoopSMS and HadoopFTS are created and deployed in the UNICORE/X component at the UNICORE service layer. With every client request originated from UNICORE clients and authenticated by the Gateway, Web service instance reads out the distributed storage system configuration files and instantiates the abstract filesystem object with the appropriate storage system. This filesystem instance holds the necessary information to communicate with the distributed storage system and directly performs operations on the DSS. Simple storage management operations are performed by HadoopSMS directly while for data transfer it initiates the HadoopFTS instance with desired data transfer protocol. This FTS service instance transfer data between DSS and USpace.

HDFS supports multi-user with its own permissions model for files and directories, a very similar one to POSIX model. UNICORE user is required to have a user account on the cluster machine with appropriate configuration settings for authorization. A user can be a single user or a group of users which is identified with the host operating system. A number of other mechanism for authentication like Kerberos, LDAP are also under consideration for future release [14].

Currently the UniHadoop prototype is being further refined and integrated into the core UNICORE server distribution.

5 Usage Scenarios

Hadoop supports a variety of back-end file systems: Hadoop Distributed File System (HDFS), CloudStore [15], Amazon Simple Storage System (Amazon S3) [16], FTP File System, Read-only HTTP and Read-only HTTPS can be used. Using UniHadoop, these can be used seamlessly in UNICORE.

The primary use case of UniHadoop is the realisation of a large, scalable storage on commodity hardware using the Hadoop HDFS.

User may need to have multiple UNICORE sites depending upon needs and resources availability. With UniHadoop integration, user can use multiple sites to access single distributed storage system. Same configuration files are only required to realized this scenario. Fig. 2 shows two sites namely site 1 and site 2 retrieve and store data on same Hadoop supported DSS. The number of sites to shares DSS may range from 1 to N.

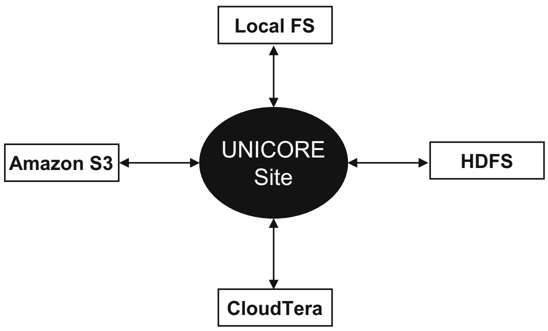


Fig. 3. Single UNICORE Site With Multiple Storage Systems

Since UniHadoop is not tied to any particular back-end, it can also be used to store data over Amazon Simple Storage System (S3) without any capacity restriction [17] and few other storage systems as shown in Fig. 3. Amazon Elastic Compute Cloud (Amazon EC2) can be used for running MapReduce jobs over data stored in Amazon S3 [18]. Data transfer among to Amazon EC2 is free which makes Amazon S3 attractive.

6 Conclusion and Outlook

In this paper, we presented UniHadoop, that represents the integration of Hadoop supported DSS with UNICORE. Now user can use the HDFS file system and some other prominent and state-of-the-art distributed storage systems as well like Amazon S3 and CloudTera with UNICORE. It enables users to administrate the previously mentioned distributed storage systems from UNICORE 6 and also to perform file transfer from these storage systems to UNICORE. A wide variety of scenarios can be realized with UniHadoop. It is possible to build highly scalable distributed storage systems using HDFS. Multiple UNICORE sites can access the same DSS to achieve load-balancing and/or high-availabity. With multiple service instances, UNICORE sites can use multiple storage systems at the same time.

Storing data using the Hadoop storage system brings a number of advantages. Primarily one can build highly scalable, efficient and reliable storage systems on commodity hardware.

As an outlook, it will be interesting to leverage the MapReduce [13] framework available in Hadoop, using HDFS for simplified parallel data processing.

Examples for using MapReduce in Hadoop are demonstrated in Pig [19], a platform for analyzing large data sets. It uses Pig Latin [20], a very powerful language for writing jobs. Hive [21], a data warehouse infrastructure built over Hadoop, can be used with data stored in Hadoop for data summarization, ad hoc queries and analysis of large set of data. It provides a simple SQL based query language, Hive QL, for simplifying queries. Theoretically UNICORE can leverages from all the projects which Hadoop supports.

The Option to run MapReduce jobs directly on the data stored in the HDFS is very attractive, for example to index large volumes of user-generated data, in order to provide powerful search and metadata capabilities.

References

1. Newman, H.B., Ellisman, M.H., Orcutt, J.A.: Dataintensive e-science frontier research. Communications of the ACM 46, 68–77 (2003)
2. Breuer, D., Erwin, D., Mallmann, D., Menday, R., Romberg, M., Sander, V., Schuller, B., Wieder, P.: Scientific Computing with UNICORE. In: Wolf, D., Münster, G., Kremer, M. (eds.) NIC Symposium 2004, Forschungszentrum Jülich; proceedings (2004)
3. Streit, A., Erwin, D., Mallmann, D., Menday, R., Rambadt, M., Riedel, M., Romberg, M., Schuller, B., Wieder, P.: Unicore - from project results to production grids. In: Grid Computing and New Frontiers of High Performance Processing (2005)
4. Foster, I., Kesselman, C., Nick, J.M., Tuecke, S.: The physiology of the grid. In: Berman, F., Hey, G.C.F.A.J.G. (eds.) Grid Computing, vol. pages 217-249, John Wiley & Sons, Chichester (2003)
5. Riedel, M., Schuller, B., Mallmann, D., Menday, R., Streit, A., Tweddell, B., Shahbaz Memon, M., Shiraz Memon, A., Demuth, B., Lippert, T., Snelling, D., van den Berghe, S., Li, V., Drescher, M., Geiger, A., Ohme, G., Vanni, A., Cacciari, C., Lanzarini, S., Malfetti, P., Benedyczak, K., Bala, P., Ratering, R., Lukichev, A.: Web services interfaces and open standards integration into the european unicore 6 grid middleware. In: Proc. Eleventh International IEEE EDOC Conference Workshop EDOC '07, October 15–16, pp. 57–60 (2007)
6. Wsrf-technical committee, http://www.oasisopen.org/committees/wsrf/
7. Unicore clients, http://www.unicore.eu/unicore/architecture/client-layer.php. (accessed on 23.05.09)
8. Soap specification, http://www.w3.org/TR/soap/
9. Web services interoperability (ws-i), http://www.ws-i.org

10. Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Gawor, J., Meder, S., Siebenlist, F.: X.509 proxy certificates for dynamic delegation (2004)
11. Howard, S.G., Gobioff, H., tak Leung, S.: The google file system (2003)
12. Apache software foundation. hadoop distributed file system, http://hadoop.apache.org/core/
13. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters (2008)
14. Hadoop pemissions, http://hadoop.apache.org/common/docs/ (accessed on 26-05-09)
15. Cloudstore, http://kosmosfs.sourceforge.net/
16. Ripeanu, M., Iamnitchi, A.: S4: A simple storage service for sciences
17. Amazon s3 with hadoop, http://wiki.apache.org/hadoop/AmazonS3 (accessed on 26-05-09)
18. Hadoop with amazon ec2, http://wiki.apache.org/hadoop/AmazonEC2 (accessed on 26-05-09)
19. Apache software foundation. pig software, http://hadoop.apache.org/pig/ (accessed on 26-05-09)
20. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing (2008)
21. Apache software foundation. hive, http://wiki.apache.org/hadoop/Hive (accessed on 26-05-09)

A Data Management System for UNICORE 6

Tobias Schlauch[1], Anastasia Eifer[1],
Thomas Soddemann[2], and Andreas Schreiber[1]

[1] Simulation and Software Technology
German Aerospace Center
51147 Cologne, Germany
Tobias.Schlauch@dlr.de, Anastasia.Eifer@dlr.de, Andreas.Schreiber@dlr.de
http://www.dlr.de/sc

[2] Fraunhofer Institute for Algorithms and Scientific Computing (SCAI)
Sankt Augustin, Germany
Thomas.Soddemann@scai.fraunhofer.de
http://www.scai.fraunhofer.de

Abstract. Data produced in scientific and industrial applications is growing exponentially but most resource middleware systems lack of appropriate support for data and metadata management. In particular easy and intuitive retrieval of data for later use is a serious problem.

In this context the paper proposes a pragmatic approach for data management of distributed data with focus on appropriate means for data organization improving data retrieval.

The paper presents the key concepts and architecture of a dedicated data management system for sharing data located on heterogeneous storage resources. The different specifics of storage systems such as data object names, data locations, and data access methods are abstracted to allow transparent data access. Moreover, the system provides means for data structuring and organization by supporting custom data models and annotation of individual metadata on data objects.

Current development status of the system is illustrated by presenting an integration with the UNICORE Rich Client which has been validated in the context of the AeroGrid project.

Keywords: UNICORE, DataFinder, Distributed Data Management.

1 Introduction

The amount of data handled by applications is growing and growing. Especially scientific applications (e.g., in astrophysics) are dealing with hundreds of Terabytes of data today. This trend is similar in the industrial sector, although the absolute numbers are a little lower. Most of the data originates from experiments and simulations. It is obvious that astrophysical simulation and high energy physics experiments produce vast amounts of data. At least with the advent of robust design methods (e.g., for the automotive and aerospace industries) huge amounts of simulation data are also produced by manufacturing companies.

With the increasing amount of data stored on disk and tape silos, it is more and more problematic to find and retrieve needed data sets efficiently. A solution is the annotation of the data with meaningful metadata.

The specific data structuring and metadata depends very much on the field of application. In general, the technical requirements for data management such as the ability for flexibly organizing the data, for annotation with various types of metadata, and for accessing a variety of storage resources are very similar in most applications. To support its institutes in this domain, the German Aerospace Center (DLR) has developed and deployed the data management application DataFinder [1]. DataFinder allows its users to manage data and metadata in an efficient manner.

In addition, resource management is as important as data management. Especially in times with growing energy costs, it is crucial that energy hungry compute resources are used efficiently. The AeroGrid project [2] a cooperation between industry, research centres, and universities is using the UNICORE 6 [3,4] middleware as their tool of choice for resource management. Unfortunately, today most resource management infrastructures lack a serious data and metadata management support. Thus a combination and cooperation of the two existing middleware stacks, UNICORE 6 and DataFinder, would be a good solution for the DLR and its partners. Hence, in this paper, we describe a first implementation of a dedicated data management system keeping compatibility with DataFinder and its integration in the UNICORE 6 middleware.

The paper is organized as follows. Section 2 provides an overview about data management support in selected distributed resource management middleware systems and dedicated systems suitable for distributed data management. Section 3 describes the data management system concept and the architecture in detail. The implementation including a description of the AeroGrid test bed is pointed out in Section 4. Finally, Section 5 gives a summary and describes future work.

2 Data Management–A Brief Overview

A key problem managing large amounts of data is fast and intuitive retrieval of produced data for later use. To accomplish this task, a data management system should support logical organization of data objects independently from storage resource specifics. Thus transparent data access is achieved without worrying about for instance migration tasks running in background, concrete data locations which might get broken over time, or provision of additional authentication information to gain access. Furthermore, the data management system should provide advanced means for data organization such as mapping of data object relations, metadata management, concepts for applying standard metadata sets, or access to data objects through specification of metadata search queries.

Like in most distributed resource management middleware systems, data management support in **UNICORE 6** is offered in form of simple data transfers to and between different UNICORE nodes. In addition, a storage service for accessing hierarchically organized file systems is available [5]. Nevertheless, this service does not provide means for accessing data objects independently from concrete storage locations or advanced data structuring functionalities.

The **Globus Toolkit** provides a set of Web services for data movement and data replication [6]. In this context replicas can be identified using logical file names and data transfer is performed on basis of GridFTP. Specific means for data structuring or ordering are not provided.

Basically, **gLite** Grid middleware supports organization of files in logically arranged hierarchies [7]. Furthermore, means for replica and basic metadata management are provided. However, this functionality cannot be used separately from gLite.

The data management system **dCache** [8] manages disk pools distributed on different server systems. Data access is achieved independently from concrete data location through specific dCache commands. However, dCache provides no support for metadata management.

The **Storage Resource Broker** [9] is originally developed by San Diego Super Computing Center (SDSC) and focuses on realization of data federations beyond location and organization boundaries. In SRB no central control or administration exists. Thus every organization keeps control of data of its domain. SRB supports organization of files in a global, logical name space, data replication, and basic metadata management with a metadata catalog. Now, work is focused on the Integrated Rule-Oriented Data System (iRODS) which is the official successor of SRB.

iRODS [10] is a complete rewrite of the SRB following a rule-based approach. Customization of iRODS (e.g., adoption to a specific set of data management policies) is achieved by mapping these policies to iRODS rules without touching the core system. Beginning with release 2.0 iRODS has nearly caught up with the SRB functionality and now migration efforts from SRB to iRODS in existing SRB projects are on the way. However, metadata management functionalities have conceptually not been extended.

The **Chemomentum** [11] project focuses on collection, storage, and usage of shared data and metadata. Key features of data-related services are the provision of a global, logical data view, flexible metadata management functionalities, and support of data replication. The provided services are implemented as atomic Web services and can be used independently from a specific Grid middleware. However, no separated production release of these services is available so far.

DataFinder [1]is general purpose software for data management with focus on scientific and technical data. The data is annotated with meta-information and ordered into data structures. The customization of these structures is achieved through support of free-definable data models which also facilitate standard metadata annotations of managed data objects. Thus the system provides a standardized logical view on the managed data. In addition, DataFinder allows simple workflows to be automated with scripts and can be easily extended with additional functionality to achieve integration in an existing working environment. Furthermore, DataFinder acts as a single point of access to heterogeneous data storage resources. DataFinder has been designed as a client-server system and provides rich user clients to allow usage of the basic data management functionalities encapsulated by a Python Application Programming Interface (API). Through the API the various storage resources and the metadata server are accessed. The communication with the metadata server takes place via the standardized Web-based Distributed Authoring and Versioning (WebDAV) protocol [12]. DataFinder is available as open source software under BSD license [13].

Most of the considered systems support a kind of storage virtualization (i.e., they provide means for arranging data objects independently from storage resource specifics). Additionally, some systems provide metadata management functionalities allowing simple annotations on data objects which are achieved using a metadata catalog component. Unfortunately, further means regarding data organization are missing. An exception from that rule are the data-related services developed in the Chemomentum project and the DataFinder system which are providing sophisticated means for data organization.

3 Data Management System Concept

A user expects an open standardized interface from a data management system. Unfortunately, standardization efforts in this direction have not yet gone far enough to cover our needs (Section 2). Hence, here a pragmatic approach on the basis of the DataFinder concepts has been chosen. The choice for DataFinder has been taken as it offers required means for data structuring and metadata management and its productive use in various scenarios provides a good basis demonstrating and investigating the developed system.

To provide the desired functionality, it is necessary to introduce abstractions for common data management concepts like data object names, storage resources, user and groups, and the methods for interacting with them. These abstractions hide the specifics of storage systems such as data access methods, data locations, as well as user authentication and authorization from the user. Realization of these abstractions can be achieved by defining the following logical name spaces.

1. **Logical data object names:** The system allows logical organization of data objects into hierarchies of collections and files. Additionally, the logical context of every data object can be expressed by specification of suitable metadata.
2. **Logical storage resource names:** The system identifies specific storage resource by a logical, unique name. This can for instance be used to realize transparent addition or removal of storage resources.
3. **Logical user names:** Every user is identified by a unique name within the system which enables the system to maintain access constraints for specific data objects.

By the logically organized data objects the system supports the definition of custom data models for specific parts in the logical data object name space. Thus fine-grained restriction of the logical data structure is facilitated. Moreover, every data type specifies a default metadata set to enforce default metadata annotation of data objects.

The data management system possesses a client-server architecture which consists of two principal components, as shown in Figure 1.

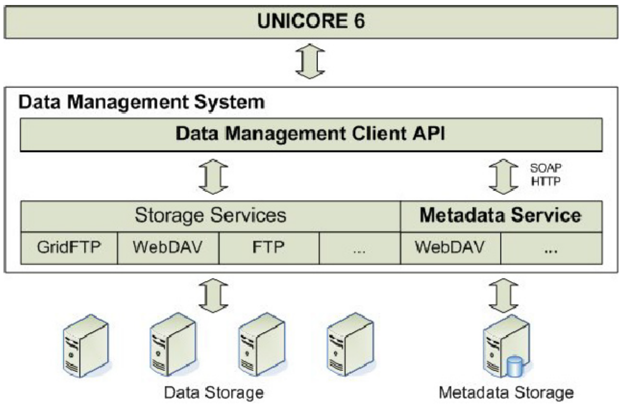


Fig. 1. Architecture of the Data Management System

Metadata Service is the core component which implements the logical name spaces. The name spaces are implemented by maintaining mappings and specific metadata in persistent metadata store. In detail the service provides file system-like functionality for data object organization within a global logical name space (e.g., creating, copying, moving, deleting data objects) and enforces restrictions of the specifically valid data model. Moreover, it provides information about data locations of data objects by maintaining global storage service configurations and links to the concrete storage locations. Furthermore, the service is intended to manage metadata and access privileges of data objects. Additionally, administrative functionality for maintenance of custom data models and storage service configurations are provided.

Data Management Client API uses the underlying storage services and the metadata service to provide data access through a uniform interface. By this API integration with UNICORE as well as other Grid middleware or software systems can be achieved. Basically, the client exports the functionalities provided by the metadata service (e.g., manipulation of logical structure or metadata access). Additionally, the client handles data access (i.e., it queries the metadata service for concrete data locations and initiates data transfers). In this context it is intended to bundle the data management client with existing standardized data transport interfaces to support interoperability and to reuse existing storage infrastructures.

4 Implementation

A first version of the data management client API bundled with data access over WebDAV and the metadata service component using a WebDAV server as storage backend have been implemented. The service fully implements all described functionalities except the access privilege management. Moreover, a plug-in for the UNICORE Rich Client has been developed to integrate the data management system with the UNICORE 6 Grid middleware. A first test installation of the system for further investigation has been established in the AeroGrid project. The software packages are available on the project site [14].

4.1 Metadata Service Implementation

The metadata service functionalities are provided through an atomic Web service. Technically, the service is implemented in Java using the Axis2 Web service framework. The Axis2 framework [15] has been chosen because it is widely adopted, supports relevant Web service standards, and allows deployment of Web services using standard web application containers.

The service has been designed in accordance to the contract first principle, i.e., beginning with the definition of the service interface using the Web Services Description Language (WSDL) version 1.1 [16] to provide a clean and simple interface. The metadata service owns a layered architecture which separates the core functionality (e.g., creation of a specific data object) from the concrete storage backend by using a dedicated persistence layer. The usage of the persistence layer allows the interchangeability of the concrete storage backend.

The metadata service uses the WebDAV protocol for the persistence of the data objects and different configuration resources. In the first place this is required to keep compatibility with DataFinder. However, WebDAV provides functionalities for the organization of files and collections in hierarchies and for the management of custom metadata that can be directly used implementing base concepts of the metadata service. Furthermore, the WebDAV extension *Access Control Protocol* [17] defines mechanisms which would allow an easy extension of the metadata service by access privilege management.

Data Object Name Space. The service manages data objects within a logical name space which is exemplarily illustrated in Figure 2.

The data object name space is organized hierarchically and starts with a virtual root data object. Every data object is associated with a logical name and is clearly addressed by its path (e.g., */Application 1/TRACE/Geometry*). In the hierarchy collections and files are distinguished. Collections can contain further collections and files. Files comply with leafs in the hierarchy and thus contain no further data objects.

Collections below the root data object correspond to a specific data management application (i.e., a specific configuration consisting of a data model and a set of storage service configurations are specifiable for these applications). This allows seamless management of data from different domains which implies different data relations.

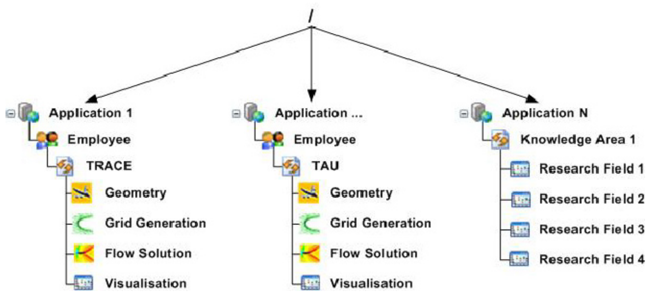


Fig. 2. Exemplary Data Object Name Space

Data Organization. In this context a data model consists of hierarchically arranged data type definitions beginning with a virtual root data type. This hierarchy can be seen as a template for the logical structure of data objects in which every data object is associated with a specific data type. Thus a suitable data structure reflecting data object relations can be defined to help identification of relevant information. Additionally, data types define sets of default metadata to support standard metadata annotations. In this context a further distinction of mandatory and non-mandatory metadata is made. The metadata is annotated on data objects with key value pairs. Metadata values can only be expressed using simple types such as *String*, *Number*, *Date*. The metadata service enforces that structural changes are made in accordance to the data model and that mandatory metadata is provided on creation time and on succeeding metadata updates.

For each data management application a custom data model reflecting the specifics of the managed data objects can be supplied. When creating a new application a default data model is provided reflecting the file system specific entities, namely directory and file. Afterwards the data model can be adapted to the specific requirements of the application by using the administrative interface of the service.

Data Location Service. To provide the data location service the system maintains storage service configurations. These configurations define the interface used to access data and store interface specific configuration parameters. Each data management application maintains its own set of configurations.

The following illustrates the treatment of data locations for importing a file: The data management client is bundled with a specific client-side implementation for accessing data through standardized data transfer interfaces. These interfaces and additional parameters are referenced in the managed storage service configurations. When creating a file data object the metadata service requires the specification of the storage service storing the data. By this information the service determines the concrete location and returns a Uniform Resource Locator (URI) [18] identifying the data location. For later data access the metadata service manages a data location link in the data object‘s metadata.

4.2 Data Management Client for UNICORE

The basis of the client is the data management client API which allows access to the above described functionalities of the metadata service and data transfer using WebDAV. The client is implemented as an Eclipse plug-in with extensions which permit a seamless communication with the UNICORE 6 Eclipse based client application.

Its graphical user interface (GUI) design is very similar to the existing DataFinder client. It allows users to browse and manipulate data object structure. Metadata can be manipulated under the constraints given by the active data model. Integration with the UNICORE 6 client is achieved by allowing users to select input and output files in the usual file browser fashion.

The implementation of the client is done by inheriting the model view controller pattern. The controller receives all events which originates from the GUI itself or UNICORE 6 directly. By using the model and GUI components, it triggers appropriate actions such as transferring data and metadata.

4.3 AeroGrid Test Bed

In the AeroGrid project data resulting from turbo machinery simulation is managed with DataFinder. In particular DataFinder has been extended to automate the aspects of simulation workflow execution. Figure 3 shows the AeroGrid deployment scenario.

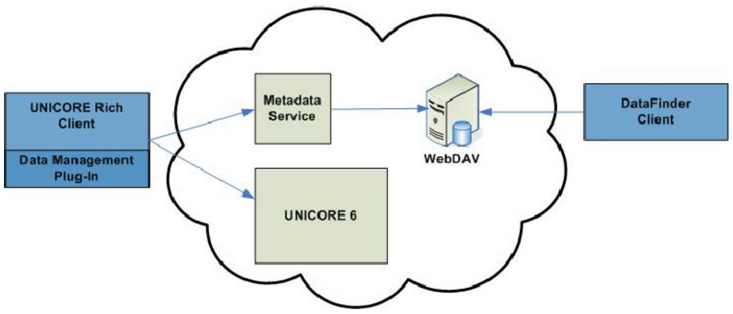


Fig. 3. Deployment Scenario of AeroGrid

The metadata service component has been deployed in the AeroGrid infrastructure independently from UNICORE 6. Both components, the metadata service and the DataFinder, are using the WebDAV server as metadata storage backend. In this context data is stored on this WebDAV server as well. The user is able to access via the UNICORE 6 Client and the developed plug-in logical data structures already managed with DataFinder through the metadata service. Data transfers can be performed identifying data locations with the help of the metadata service and using the bundled WebDAV data transfer interface. Thus simulation runs can be initiated using UNICORE Rich Client and DataFinder.

The compatibility of the developed data management system with DataFinder is basically achieved on level of the metadata service. Both components are using the WebDAV protocol for storage of logical data structures and configuration resources. Moreover, the metadata service keeps the DataFinder scheme for annotating data objects with metadata as well the format of configuration resources. This provides a seamless integration of the metadata service with DataFinder. Moreover, the DataFinder administration client can be used to configure data models and storage service configurations which is not yet covered by the UNICORE Rich Client plug-in.

On this basis the functionalities of the metadata service and in particular the compatibility with DataFinder have been successfully validated. Because the Web service stack is additionally involved when accessing metadata, it is expected the implemented system causes specific overhead in comparison to DataFinder. Therefore, an advanced deployment scenario suitable for performance assessments and comparisons to DataFinder will be established.

5 Summary and Outlook

In the first part of the paper an overview about data management support in selected resource middleware systems and dedicated systems suitable for distributed data management has been given. The considered systems provide a kind of storage virtualization but no sophisticated means for data organization. Concerning growing amounts of data produced by current and future applications, data retrieval for later use becomes a serious problem.

Thus the paper proposes a pragmatic approach for a dedicated data management system based on DataFinder. The system delivers logical organization of data objects and abstracts common data management concepts hiding the specifics of storage systems. On this basis advanced means for data organization through metadata management functionalities and support of custom data models are provided. Moreover, the integration with UNICORE 6 is achieved by the data management client API which has been integrated in the UNICORE Rich Client. The developed system and its compatibility with DataFinder have been successfully validated in the AeroGrid project.

The next step concerns a detailed investigation of system performance which will be made in the AeroGrid project. Moreover, additional features are intended to be developed. This concerns the provision of fine-grained access privilege management by *Access Control Lists* and the implementation of a data object retrieval operation by metadata search queries.

Acknowledgments. This work has been partly supported by the German Federal Ministry for Research and Technology (BMBF) under Grant 01IG07006.

References

1. Schlauch, T., Schreiber, A.: Datafinder – a scientific data management solution. In: Ensuring the Long-Term Preservation and Value Adding to Scientific and Technical Data, PV 2007, Oberpfaffenhofen, Germany (2007)
2. German Aerospace Center (DLR): AeroGrid website, `http://www.aero-grid.de`
3. Erwin, D., Rambadt, M., Streit, A., Wieder, P.: Production-quality grid environments with unicore. In: Wallom, D., Kielmann, T. (eds.) Proceedings of the Workshop on Grid Applications: From Early Adopters to Mainstream Users, Global Grid Forum, pp. 10–17 (2006)
4. Jülich Supercomputing Centre: Unicore, `http://www.unicore.eu/unicore`
5. Jülich Supercomputing Centre: Unicore wiki - extending sms, `http://unicore.wiki.sourceforge.net/ExtendingSMS`
6. globus.org: Data management: Key concepts, `http://www.globus.org/toolkit/docs/4.0/data/key/`
7. Enabling Grids for E-Science (EGEE): EGEE gLite User‘s Guide - Overview of Glite Data Management (March 2005)
8. Fuhrmann, P., Gülzow, V.: dcache, storage system for the future. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1106–1113. Springer, Heidelberg (2006)
9. Baru, C., Moore, R., Rajasekar, A., Wan, M.: The sdsc storage resource broker. In: Proceedings of CASCON (1998)
10. Rajasekar, A., Wan, M., Moore, R., Schroeder, W.: A prototype rule-based distributed data management system. In: HPDC workshop on Next Generation Distributed Data Management (2006)
11. Schuller, B., Demuth, B., Mix, H., Rasch, K., Romberg, M., Sild, S., Maran, U., Bata, 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)
12. Goland, Y., Whitehead, E., Faizi, A., Carter, S.R., Jensen, D.: Http extensions for distributed authoring - webdav (February 1999), `http://www.ietf.org/rfc/rfc2518.txt`
13. German Aerospace Center (DLR): DataFinder, `http://datafinder.sourceforge.net`
14. Fraunhofer Institute for Algorithms and Scientific Computing (SCAI): Unicore-datafinder project, `http://tor-2.scai.fraunhofer.de/gf/project/unicoredata/`
15. The Apache Software Foundation: Apache axis2/java - next generation web services, `http://ws.apache.org/axis2/`
16. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (WSDL) 1.1, `http://www.w3.org/TR/wsdl`
17. Clemm, G., Reschke, J., Sedlar, E., Whitehead, J.: Web distributed authoring and versioning (webdav) - access control protocol (May 2004), `http://www.ietf.org/rfc/rfc3744.txt`
18. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform resource identifier (uri): Generic syntax (January 2005), `http://www.ietf.org/rfc/rfc3986.txt`

UNICORE-Related Projects for Deploying the Belarusian National Grid Network

Andrew Lukoshko and Andrei Sokol

Republican Supercomputer Centre, United Institute of Informatics Problems,
National academy of Sciences of Belarus, Surganova str., 6, 220012 Minsk, Belarus
{Andrew.Lukoshko,Andrei.Sokol}@gmail.com

Abstract. UNICORE middleware was chosen for the deployment of the National Grid network of Belarus. For quick and easy UNICORE installation on Grid sites the specialized distribution was developed. It includes not only UNICORE services and clients, but some other components such as MPI libraries and Torque batch system. At the beginning of 2009, there was launched the development of a billing system in order to monitor the employment of National Grid resources. Billing system provides various accounting information from Grid sites through Web-interface.

Keywords: Grid Computing, SKIF-GRID, UNICORE, distribution, billing.

1 Introduction

The Scientific and research program of The Union of Russia and Belarus -"The Development and Use of Software and Hardware Grid technologies of the prospective high-performance (supercomputer) computer systems "SKIF" (code "SKIF-GRID"[1]) for the period of 2007-2010, was developed in order to implement the decree of the Board of Ministers of the Union d/d June 28th, 2006 [1].

One of the goals of the project is to create National Grid networks in Belarus and Russia using existing supercomputer centers, as well as participation in international projects.

Russia and Belarus went separate ways in the deployment of National Grid networks. In the year 2007, the main Grid middleware products were analyzed: UNICORE, gLite, Globus Toolkit, X-COM (the Russian project) and Condor-G. UNICORE [2] was selected to deploy the Belarusian National Grid network, and in 2008, the UNICORE-based distribution was developed in The United Institute of Informatics Problems at The National Academy of Sciences of Belarus [3]. We note "SKIF-GRID" program also stipulate integration various grid-segments based on gLite middleware with pan-European grid nets in framework of BalticGrid-II Project[2].

[1] Work is supporting financially by Joint Belarusian-Russian Program "SKIF-GRID" and performing under the contract to the National Academy of Sciences of Belarus, Minsk.

[2] Work is supporting financially by EU and performing under the contract 223807 to European Commission, Brussels.

At the beginning of 2009, there was launched the development of a billing system in order to monitor the employment of National Grid resources. In future, using of the National Grid will become a paid service, and a global billing system will be the only way of accounting.

This paper describes the UNICORE distribution and billing system for the National Grid Network in Belarus.

2 Custom UNICORE-Based Distribution

UNICORE (UNiform Interface to COmputing REsources) was chosen for the deployment of the national Grid network of Belarus among the variety of middleware. The package complies with customary requirements of free-of-charge basis and open source, and has some other characteristics that made it the premier choice. It's cross-platform, small, simple to deploy and administer (just a few basic services, each has strictly defined set of functions), constantly evolved and accurately maintained.

The goal of this project was to develop a UNICORE-based software product, which includes installation packages and a complete set of technical documentation.

The following operating systems were selected for deploying Belarusian Grid sites:

- Debian GNU/Linux 4.0 x86_64.
- Fedora 8 x86_64.
- Scientific Linux 4 i386.
- Windows XP 32 bit.

For quick and easy deployment of Grid sites the distribution includes not only UNICORE services and clients, but some other components (Fig. 1). For Linux systems these are the following:

- Torque batch system.
- Java Runtime Environment.
- OpenMPI.

Respectively, distribution for Windows includes:

- MPICH.
- Java Runtime Environment.
- MinGW & MSYS (GNU utilities and GCC compiler).

For Linux operating systems distribution components come in native formats - DEB and RPM. As for the Microsoft Windows, it was decided to create a custom NSIS-based installer, designed specifically for this platform (Fig. 2).

Another feature of the Belarusian UNICORE distribution is a specially designed *configure.py* configuration tool's analogue for Windows. This script has been rewritten using the PHP and the language interpreter was included in the distribution. The usage of this script is completely similar to *configure.py.*

The distribution is supplemented by detailed step-by-step manuals for users (40 pages) and administrators (80 pages).

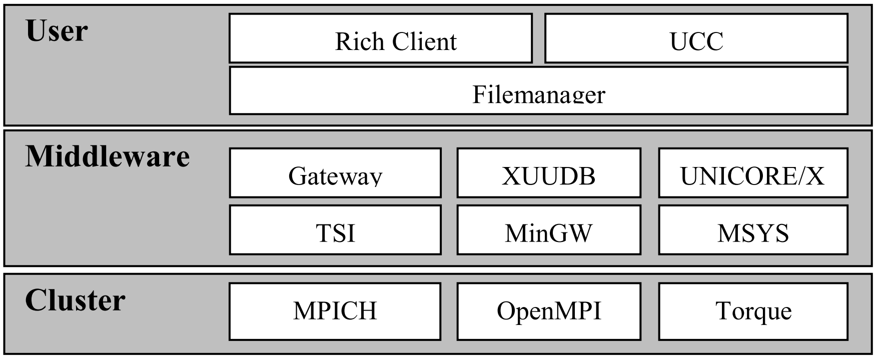
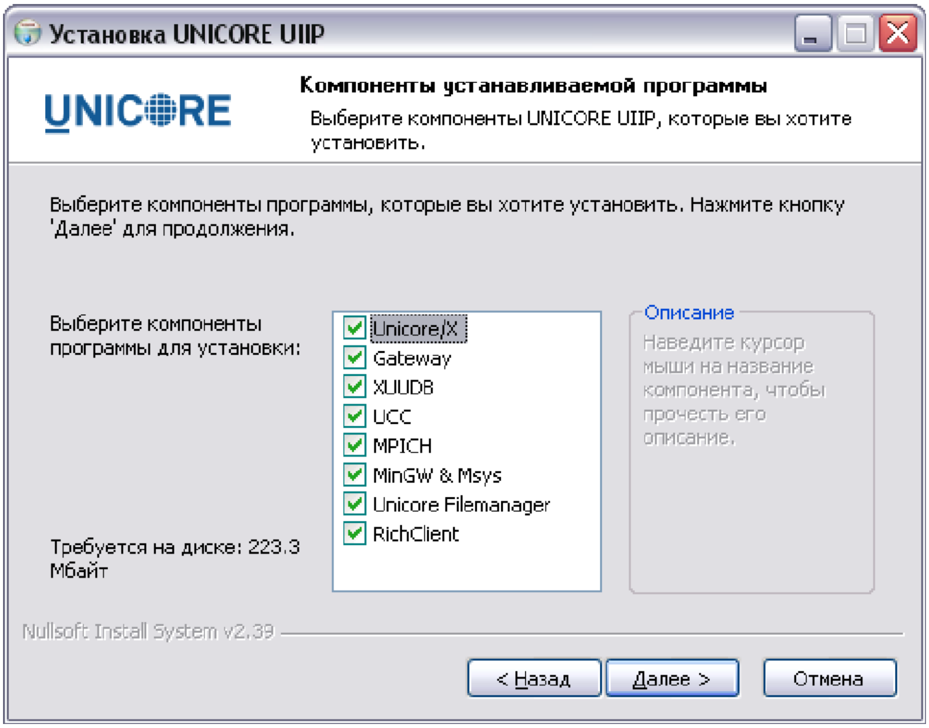


Fig. 1. UNICORE distribution's components



Компоненты устанавливаемой программы

Выберите компоненты UNICORE UIIP, которые вы хотите установить.

Выберите компоненты программы, которые вы хотите установить. Нажмите кнопку 'Далее' для продолжения.

Выберите компоненты программы для установки:

- ✓ Unicore/X
- ✓ Gateway
- ✓ XUUDB
- ✓ UCC
- ✓ MPICH
- ✓ MinGW & Msys
- ✓ Unicore Filemanager
- ✓ RichClient

Описание

Наведите курсор мыши на название компонента, чтобы прочесть его описание.

Требуется на диске: 223.3 Мбайт

Fig. 2. NSIS-based UNICORE installer for Windows

3 UNICORE Billing System

One of the current projects to be developed at UIIP National Academy of Sciences of Belarus is a billing system for national Grid network. The requirements for the product are the following:

- Recording of the actual time of computing resources usage in normalized units under SPEC2000.
- Ability to identify commercial software usage.
- Interaction with the system for different categories of users and providing relevant information.
- Various text and graphics reports about resources used by individual user, site, etc.
- Generating bills based on information above.
- All connections are secured with user and service certificates.
- E-mail notifications.

Due to National Academy of Sciences of Belarus requirements and special technical features, billing system was developed as a standalone web-based application. It consists of following components (Fig. 3):

- Application server.
- Database server.
- Agents.

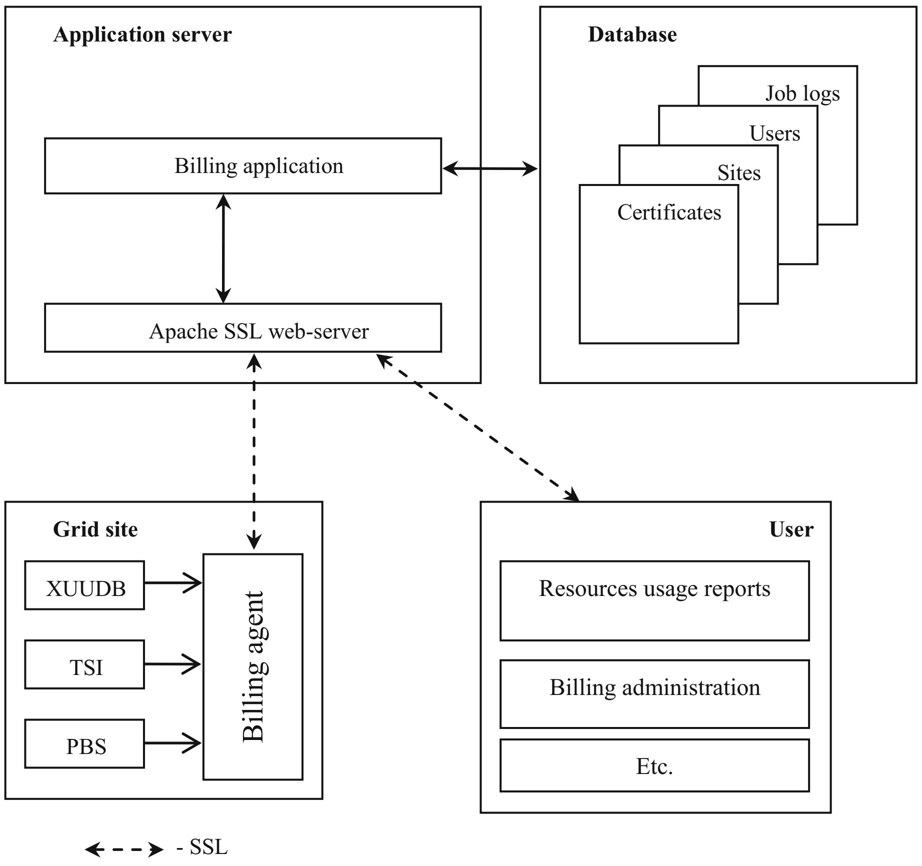


Fig. 3. Billing system schema

3.1 Application Server

Application server is based on Apache httpd server with mod_ssl and mod_php modules included. This solution makes it possible to run billing service under Linux or Windows systems. Application server is a main component with passive behavior. It doesn't make any network connections. It only responds to two kinds of requests:

1. Client's requests.
2. Billing agents' requests.

The both communicate with application server using HTTPS protocol.

3.1.1 Authentication

Any person that wishes to use a UNICORE Grid has to identify him/herself by means of a UNICORE user certificate, which is a standard X.509 certificate issued on that person's name and affiliation by a certification authority that is trusted by the sites in the Grid [4].

Billing application's main authorization unit is user, which is described by integer positive number called UserId. Each user can be assigned to multiple UNICORE user certificates. However, only the one, valid cert can be used to access billing interface. Mod_ssl was configured in a way, that web-server verify the certificate's depth (SSLVerifyDepth directive).

Table 1. Possible user's roles

User role	Comments
Grid user	— Views detailed statistics and resources usage reports and billing information; — Receives email notifications.
User's group administrator	Views detailed statistics and resources usage reports and billing information for each member of the group
UNICORE site's operator	Views detailed statistics and resources usage reports and billing information for specified Grid site
Grid operator	Views detailed statistics and resources usage reports and billing information for specified Grid sites or whole Grid
Billing service administrator	— Views detailed statistics and resources usage reports and billing information for specified users or user groups, specified Grid site(s) or whole Grid; — Manages Grid sites, agents; — Manages user, user groups; — Manages certificates; — Manages price table.

Additionally, password is used to prevent unauthorized access to billing from user web-browser with imported certificate.

3.1.2 Authorization

Authorization was implemented using Role-Based Access Control (RBAC) schema, which provides a simple yet powerful centralized access control [5].

The billing system supports the following five user's roles:

1. Grid user.
2. User group's administrator.
3. UNICORE site's operator.
4. Whole Grid's operator.
5. Billing service administrator.

User could be assigned to multiple roles at the same time.

3.2 Database Server

MySQL 5.x is used as database server for billing application. Data tables uses InnoDB storage engine. Performance tune was done, but no extra patches were applied. To avoid data loose binary logs and everyday backups to external storage was set up. For best performance database server is set up on dedicated physical server.

3.3 Agents

Agent is used to extract billing information from different sources (e. g. PBS accounting files or license server log files), match it with XUUDB user record and upload results to billing server, which accepts definite format of sent data. It is possible to develop a custom agent's version for any source, by the bringing information into definite state. Each record of sent data contains following ever-present pieces of information:

- Task id.
- Certificate id.
- Task termination time.
- Used resources.

Data is uploaded via HTTPS request. Each agent had it own certificate, signed with root UNICORE CA. Agent's certificate should be saved on billing server, otherwise billing wouldn't recognize agent and connection will be rejected.

Now we implement four kinds of agents:

1. PBS accounting agent.
2. MAUI accounting agent.
3. LS-DYNA license agent.
4. ANSYS license agent.

The first and the second installed on every Grid site and analyze information about resources used while task was computing. The others analyze license server logs and fix license usage time.

All agents are written in PHP using libcurl and gzlib. It makes possible to run agents on both Windows and Linux systems natively.

Data transmission held in four steps:

1. Agent connects to application server via HTTPS.
2. Server recognizes agent with its certificate.
3. Server finds out the last event date stored in the database according to the agent and sends it back.
4. Agent parses logs starting with received date and uploads pieces of information to server.

3.3.1 TSI Modification

Usual realization of UNICORE supposes the system login presence for each UNICORE user. On the other hand, it possible to use one system login for multiple UNICORE users. It makes Grid site's administration process simpler, but raises the question of identifying task owner. PBS stores information about system login and PBS task id, but UNICORE stores information about UNICORE user and UNICORE task id. XUUDB data can be exported into csv file, but it was mentioned above, that a lot of UNICORE users can be assigned to one system login. That is why it is impossible to find relations between PBS and UNICORE task id without any UNICORE modification. Belarusian National Grid network meets the last way of managing user accounts.

The part of TSI responsible for executing submitting command was improved in the following way. PBS submit command returns to TSI the PBS id of submitted task, at the same time UNICORE task's id and UNICORE user's info is stored in TSI's environment. This information is extracted and logged to the text file, which is parsed by billing agent. In the result, information with correct correspondences is sent to billing server.

4 Conclusion

In this paper we presented the following UNICORE-related projects for deploying the Belarusian National Grid network:

- Custom UNICORE distribution for Linux and Windows-based sites.
- Billing system for National Grid.

The technical details of each project were highlighted.

The current version of billing due to technical requirements had been implemented as a third party web application written on PHP, so it cannot be one of the UNICORE parts. The future of this project is to implement the billing system as UNICORE official add-on using obtained experience with the full developing guidelines compliance.

In reference to custom UNICORE distribution, it is production-ready and used to deploy all of the National Grid sites.

References

1. Ablameyko, S., Anishchanka, U., Krishtofik, A., Tchij, O.: Belarusian National Grid-Initiative. In: Cracow Grid workshop'08, Kraków, Poland, October 13-15 (2008), `http://www.cyfronet.krakow.pl/cgw08/prezentacje/35-Poster-CGW08.jpg`
2. UNICORE middleware, `http://www.unicore.eu`
3. Ablameyko, S., Anishchanka, U., Krishtofik, A., Tchij, O.: Setting up SKIF-UNICORE experimental Grid section. In: Cracow Grid workshop'08, Kraków, Poland, October 13-15 (2008), `http://www.cyfronet.krakow.pl/cgw08/prezentacje/40-Poster-CGW08.jpg`
4. Erwin, D.: UNICORE Plus Final Report - Uniform Interface to Computing Recources. Forschungszentrum Julich (2003)
5. Role-based access control, `http://en.wikipedia.org/wiki/Role-based_access_control`

Fourth Workshop on Virtualization in High - Performance Cloud Computing (VHPC 2009)

VHPC 2009: 4th Workshop on Virtualization in High-Performance Cloud Computing

Michael Alexander[1] and Marcus Hardt[2]

[1] Scaledinfra technologies GmbH, Vienna, Austria
[2] Forschungszentrum Karlsruhe, Germany

Virtualization has become a common abstraction layer in modern data centers, enabling resource owners to manage complex infrastructure independently of their applications. Conjointly virtualization is becoming a driving technology for a manifold of industry grade IT services. Piloted by the Amazon Elastic Computing Cloud services, the cloud concept includes the notion of a separation between resource owners and users, adding services such as hosted application frameworks and queuing. Utilizing the same infrastructure, clouds carry significant potential for use in high-performance scientific computing. The ability of clouds to provide for requests and releases of vast computing resource dynamically and close to the marginal cost of providing the services is unprecedented in the history of scientific and commercial computing.

Distributed computing concepts that leverage federated resource access are popular within the grid community, but have not seen previously desired deployed levels so far. Also, many of the scientific datacenters have not adopted virtualization or cloud concepts yet. This workshop aims to bring together industrial providers with the scientific community in order to foster discussion, collaboration and mutual exchange of knowledge and experience.

This year's workshop featured 9 papers on diverse topics in HPC virtualization. Papers of note include Checconi et al. examining QoS in VM migration times using a stochastic model along with Nanos and Koziris presenting a native I/O driver framework for Myrinet 10G network interfaces in Xen. Two papers were examining private and public cloud suitability for scientific computing.

The chairs would like to thank the Euro-Par organizers and the members of the program committee along with the speakers and attendees, whose interaction contributed to a stimulating environment. VHPC is planning to continue the successful co-location with Euro-Par in 2010.

SSD-HDD-Hybrid Virtual Disk in Consolidated Environments*

Heeseung Jo, Youngjin Kwon, Hwanju Kim, Euiseong Seo, Joonwon Lee, and Seungryoul Maeng

Korea Advanced Institute of Science and Technology (KAIST),
335 Gwahangno, Yuseong-gu, Daejeon, Korea
Ulsan National Institute of Science and Technology (UNIST),
100 Banyeon-ri, Eonyang-eup, Ulju-gun, Ulsan, Korea
Sungkyunkwan university,
300 Cheoncheon-dong, Jangan-gu, Suwon, Gyeonggi-do, Korea
{heesn,yjkwon,hjukim,maeng}@camars.kaist.ac.kr,
euiseong@unist.ac.kr, joonwon@skku.edu

Abstract. With the prevalence of multi-core processors and cloud computing, the server consolidation using virtualization has increasingly expanded its territory, and the degree of consolidation has also become higher. As a large number of virtual machines individually require their own disks, the storage capacity of a data center could be exceeded. To address this problem, copy-on-write storage systems allow virtual machines to initially share a template disk image. This paper proposes a hybrid copy-on-write storage system that combines solid-state disks and hard disk drives for consolidated environments. In order to take advantage of both devices, the proposed scheme places a read-only template disk image on a solid-state disk, while write operations are isolated to the hard disk drive. In this hybrid architecture, the disk I/O performance benefits from the fast read access of the solid-state disk, especially for random reads, precluding write operations from the degrading flash memory performance. We show that the hybrid virtual disk, in terms of performance and cost, is more effective than the pure copy-on-write disks for a highly consolidated system.

Keywords: Consolidation, Virtual machine (VM), Copy-on-write (CoW), Hybrid storage.

1 Introduction

Virtualization enables multiple operating systems to run on a single physical machine, and server consolidation systems using virtualization have expanded

* This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. 2009-0080381) and the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) Support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2009-C1090-0902-0020).

their territory significantly, especially in large-scale computing systems or cluster systems. This trend is based on the effort to lower the management cost, which is one of the primary factors for server hosting centers or the server market. With server consolidation, fewer physical machines are needed to run the same number of servers, thus saving power and space. These factors are directly related to the total cost of ownership; it is known that 50-70% of reduction could be possible [1]. Moreover, the virtualized system is also advantageous due to the availability and manageability of servers.

Storage virtualization has been less focused than other resources, such as memory and CPU, since disks have better density and are easily sharable via network attached storage. The introduction of cloud computing, however, makes efficient storage virtualization more relevant in terms of disk capacity. Cloud environments allow thousands of cloud users to store their own contents and privately view their storage. With more virtual machines (VMs), one VM will require more storage space due to operating systems and applications that become richer and larger. Therefore, the capacity requirements for storages are expected to grow exponentially. Data centers serving a large-scale VM farm cannot extend their storage infinitely, since the cost of doing so is not inexpensive, after taking into account ownership costs such as maintenance, cooling, and space. Since traditional sharing-based storage cannot deal with this requirement, many data centers are unable to afford the storage capacity for private disks required by cloud users.

Two representative approaches have been developed to relieve the burst requirement of storage in virtualized environments: copy-on-write (CoW) storage and content addressable storage (CAS) [2,3]. First, CoW storage enables multiple VMs to initially share a template disk image. This mechanism allows read-only sharing by isolating any write attempts from the template disk image. This approach was adopted in QCOW [4], CoWNFS [5], and Parallax [6]. Second, CAS uses a content-based address to access a disk block. This mechanism does not require even template disk image sharing, but incurs computational overheads. Although these two approaches significantly reduce disk footprints, they do not improve the disk I/O performances of those mechanisms.

This paper presents a hybrid CoW virtual disk that combines the solid-state disk (SSD) and hard disk drive (HDD) within a highly consolidated system. The SSD-HDD-hybrid virtual disk (HVD) uses SSD for read-only template storage, whereas privately written data are stored in HDD. HVD gains high disk I/O performance from the fast read operations of SSD, especially for random reads. Since the read operations of consolidated VMs are multiplexed, a sequential read stream of each VM could be broken, and thereby realized as small random reads. Further, the isolation of write operations from SSD eliminates drawbacks from write I/O, such as erase-before-write and wear-out. Our evaluation results indicate that the hybrid architecture of HVD outperforms HDD-only or SDD-only storage. For several real workloads, HVD shows more than 40% performance enhancement and does not suffer from the heavy write, which is the main weakness of SSD.

The rest of this paper is organized as follows: Section 2 describes the design and implementation of HVD and discusses related challenging issues. Section 3 presents the evaluation results of HVD compared with pure storage by using several micro-benchmarks and real workloads. Finally, we summarize the paper and present a future direction in Section 4.

2 Hybrid Virtual Disk (HVD)

This section describes the overall architecture of HVD. First, we give a brief description of the virtualized environments using CoW storage. Then, we present the hybrid architecture of HVD and its implementation. Finally, we illustrate migrating data between SSD and HDD, a challenging issue for HVD architecture.

2.1 CoW Storage in Virtualized Systems

In virtualized environments, the CoW mechanism over virtual disks has been prevalent due to its efficient use of disk and easy management of snapshot [12,13,6]. The CoW mechanism is a well-known technique that allows multiple entities to share a resource, until a write attempt occurs to the shared resource; once written, the shared resource is copied to a newly allocated space for the private use of the resource. In this manner, a CoW disk enables multiple VMs to share a template disk image while presenting each VM with the private view of its own storage. In addition, the CoW disk can support fast snapshots by preserving metadata for the current disk image.

The CoW disk is compelling in consolidated environments for three reasons. First, many VMs typically run the same operating systems and applications, especially in cluster-based systems, which provide replicated services for reliability and load balancing [5]. In this system, multiple VMs can share a template disk image that contains common operating systems and applications in a CoW manner, thereby reducing disk footprints. Second, as the degree of consolidation has grown considerably, a virtualized data center could accommodate many more servers than a native data center. Since each server at least requires a system image from which to boot, a large number of servers may exceed the storage capacity [14]. The CoW disk can effectively relieve this increased requirement of storage capacity. Finally, snapshot is a frequent operation used to control the history of a virtual disk for reliability. As cloud computing has emerged in large-scale consolidated environments, reliability is now a more important concern to cloud users. The efficient snapshot functionality of the CoW disk enables fast backup and recovery of storage.

2.2 SSD-HDD-Hybrid Design

To maximize the advantages and to minimize the drawbacks of SSD, we introduce HVD for virtualized environments. In HVD, the read-only templates of VM disk images are stored on SSD to support fast read operations. On the other hand, the privately written blocks of a VM are placed on HDD. This design is inspired by the asymmetric I/O characteristic of SSD; the write operation is slow and varied, whereas the read operation is fast and uniform.

SSD is currently an emerging storage device for server systems to enhance the disk I/O performance [9,10]. SSD is a NAND flash memory-based storage device that is expected to replace HDD in the near future because of its versatile features, such as non-volatility, solid-state reliability, low power consumption, shock resistance, and high cell densities [7,8,11]. SSD supports high read performance, especially for random reads, since it does not include HDD-like mechanical parts that incur seek and rotational delays. SSD, however, has several weak points caused by the nature of NAND flash memory. One is the *erase-before-write* characteristic that a page, which is the basic unit of read and write operations, should be erased before being rewritten in the same location. The erase operations can only be performed on a block, which is larger than a page. Therefore, SSD shows slow and non-uniform write latency. Another limitation is the wear-out problem. Unfortunately, each block in flash memory has a limited number of erase/write cycles, and data in a block become unreliable if the block reaches this limit. The current limit for single-level cell NAND flash memory is approximately 100,000 erase/write cycles.

Considering these features of SSD, the SSD-HDD-hybrid scheme has several advantages. First, HVD supports fast read accesses to a template disk image, which typically contains rich applications, libraries, and common data contents. HVD improves user experiences by boosting the startup of applications and the loading of libraries. In addition, random read accesses to a template disk image benefit from SSD. Since multiple sequential read streams from guest VMs are fairly multiplexed, each stream might be broken into small random read operations, which result in the poor performance of HDD. SSD provides better latency for the broken random reads. Next, isolating writes from SSD eliminates the aforementioned problems induced from write operations. As HVD preserves a template disk image on SSD from write operations, SSD does not suffer from wear-out and overheads for erase-before-write. Finally, HVD allows for cost-effective storage, in terms of performance and capacity. Since SSD is more expensive than HDD with the same capacity, pure SSD-based storage might not be an affordable option to store large amounts of private data of VMs. HVD requires SSD capacity only for template disk images, making our approach more cost-effective.

2.3 Implementation

We implemented two versions of HVD: HVD based on *cowloop* [15] and HVD based on Parallax. Our approach to hybridizing SSD and HDD for a CoW block device can be applied with low reengineering costs. Moreover, our approach is also advantageous in terms of transparency. It can be provided to upper layers without any modifications due to block level implementation.

Cowloop is a simple and lightweight block device used to support the CoW behavior. Figure 1 shows the HVD implementation overview based on cowloop. A

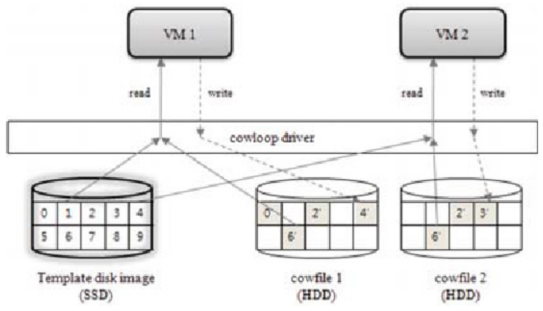


Fig. 1. The HVD implementation overview based on cowloop

VM uses the template disk image as read-only, and when the VM updates blocks, the write operations are forwarded to its cowfile, which stores the privately written blocks. If a block is written once, the next access to the block is forwarded to its cowfile. For example, block 1 of VM1 is read from a template disk image, and block 6 that is written before is read from the cowfile1. For HVD, we place the template disk image on SSD, and use HDD as cowfile storage.

On the other hand, Parallax is a novel distributed storage system for Xen VMs [20] and supports the CoW mechanism to reduce the required storage size. Furthermore, Parallax provides many features, such as network access, snapshot, and the efficient lock mechanism. Our Parallax version of HVD spontaneously inherits these features. To support the CoW behavior, Parallax uses a radix tree that translates the logical block number (LBN) from a VM to the physical block number (PBN). If a VM updates a block, the related radix tree nodes are created, and their leaf node possesses the PBN. In addition to PBN, the entry of a leaf node indicates whether a data block is read-only or written via a bit flag. For HVD, we add a 1 bit locator flag that denotes whether a block resides in SSD or HDD.

2.4 Migration between SSD and HDD

The current placement policy of HVD has optimization chances to migrate data between SSD and HDD. In cases where a file is first modified and frequently read afterward, this file is obtained from HDD without the benefit of SSD. Such write-once read-many blocks can be migrated to SSD so that better read performance is achieved. There are various possible methods to identify migratable blocks at different levels of hierarchy.

First, users can specify rules that reflect their preferences. For example, many configuration files or static web contents (e.g. /etc, html, or web image files) are initially modified and primarily read for the rest of their lifetimes. In this case, a user can define that such files should always reside in SSD. This rule-based approach should collaborate with the file system to inform HVD of blocks in which a specified file is located. While requiring user intervention, this method can directly write a specified file to SSD without migration.

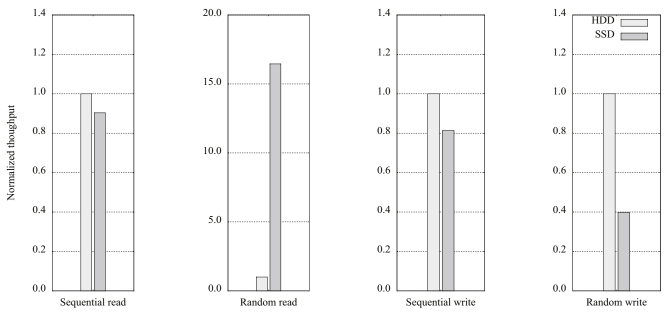


Fig. 2. The raw I/O performance comparison between HDD and SSD

Second, the file system can identify write-once read-many files by monitoring modification and access times stored in the metadata. This monitoring-based method enables frequently read files, after being written, to be migrated to SSD without user intervention. This method, however, requires a monitoring daemon in each guest VM.

Third, HVD maintains read access frequency for each block stored once in a location of HDD during a certain period. When detecting a frequently read block, HVD migrates this block to SSD. This method is guest VM-agnostic, so that no guest-level daemon is required. On the other hand, the block-level approach redundantly manages metadata for each block in order to maintain access frequency.

3 Evaluation

In this section, we evaluate the performance aspect of HVD. Our storage system is implemented on Xen-3.2.3 with a para-virtualized Linux 2.6.18 kernel for the x86 architecture. The machine for Xen has an Intel Core2 Duo 2.33 GHz CPU with 2 GB of RAM. The memory size of a driver VM, which is in charge of I/O device accesses and contains HVD, is configured to 512 MB, and that of each guest VM is set to 128 MB. All tests are performed on a local storage to exclude network overhead.

In all evaluations, we used the cowloop version of HVD, since Parallax has several functions including a garbage collector and a locking mechanism in addition to the CoW features. Although Parallax is a more sophisticated virtual disk, we suppose that the cowloop version of HVD clearly shows the performance gain from our hybrid approach to exclude the impact of additional features, except the CoW mechanism. To demonstrate the impact of our hybrid approach, we evaluate HVD in comparison with the pure CoW disks: cowloop-HDD and cowloop-SSD.

Raw device performance. Seagate barracuda with 7200 RPM [18] and Samsung SSD [19] are used in all evaluations. These storage devices are selected for

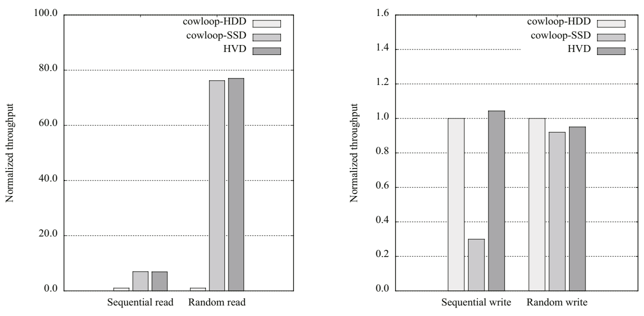


Fig. 3. The I/O performance of cowloop-HDD, cowloop-SSD, and HVD for each operation type

a reasonable performance comparison. Figure 2 shows the raw performances of HDD and SSD, which are used in this evaluation for several types of disk operations (sequential read/write, random read/write), and the results are normalized to HDD. We performed the tests using *sysbench* [16] on a native machine. As depicted, except in the case of the random read, HDD performs better than SSD. In the case of the random read, however, SSD shows better performance than HDD by the multiple of sixteen.[1]

Micro-benchmark. Figure 3 shows the evaluation result of micro-benchmark using sysbench. All the tests are performed on a guest VM, and all I/O operations are delivered to each storage device through a driver VM. The tests are performed for sequential read/write and random read/write, and the y-axis shows the normalized throughput.

In the case of the read operation, SSD shows significant effects, especially for the random read. For the sequential read, unlike a native environment, both cowloop-SSD and HVD indicate higher throughput than cowloop-HDD. While HDD maximizes the sequential read performance for a burst read, a driver VM interrupts read operations, breaking burstness and thus reducing HDD read performance.

For the write operation, the performance of cowloop-HDD and HVD is similar as expected. The sequential write operation of HDD is much faster than that of SSD due to the erase-before-write characteristic of SSD. In the case of random write, HDD shows little higher performance than SSD, since HDD incurs seek and rotational overheads.

Real workloads. With regard to real workload evaluations, we performed four workloads: the booting of VMs, the online transaction processing (OLTP), the decompression, and the data writing. The first two are read-intensive workloads, and the decompression is read/write mixed with a ratio of 1.5. The last data

[1] A recent high-end SSD for server environments outperforms HDD for all disk operations, but our current experiments are not conducted with a high-end SSD for a fair comparison.

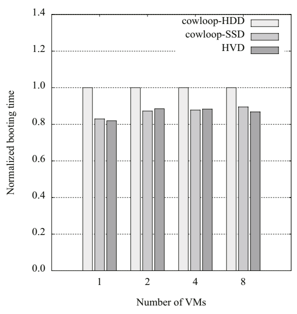


Fig. 4. The booting time of VMs

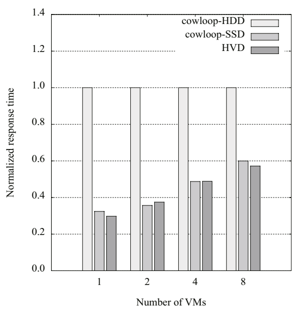


Fig. 5. The response time of OLTP

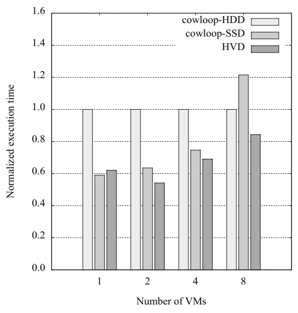


Fig. 6. The execution time of decompression

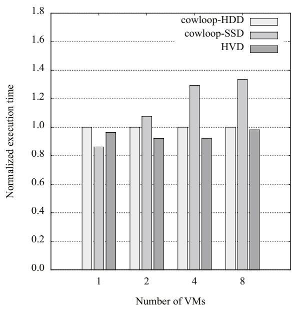


Fig. 7. The execution time of data writing

writing is a write-intensive workload. The VM configurations for all the tests are the same as that of the micro-benchmark test. We evaluate each workload as increasing the number of VMs. Figure 4 shows the normalized booting time of guest VMs. The performance gain of HVD is not considerable, since the booting sequence of a VM involves only a little amount of I/O operations. The next test is the online transaction processing with Mysql [17] and sysbench, which requests approximately 130 database transactions per second. The evaluation results are illustrated in Figure 5, and the y-axis is the normalized average response time. The response time of cowloop-SSD and HVD are 30-60% less than that of cowloop-HDD.

On the other hand, the evaluation result of the decompression is presented in Figure 6. This workload decompresses the source code of Xen and Linux. The notable situation occurs when the number of VMs is eight. The execution time of cowloop-SSD is longer than that of cowloop-HDD, since SSD shows the slowest operation latency for heavy random writes, due to erase-before-write. The same case is more clearly shown in the write-intensive workload, the data writing. As presented in Figure 7, cowloop-SSD results in a longer execution time when the number of VMs is more than one. All the results illustrate that HVD has higher disk

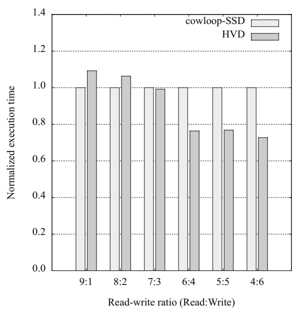


Fig. 8. The execution time of file read/write with different read-write ratios

performance than cowloop-SSD and cowloop-HDD, especially when a large number of VMs are consolidated. More significantly, outperforming pure SSD means that HVD is more cost-effective for server consolidation workloads. For more detailed analysis, we performed an additional evaluation that executes file read and write with different read-write ratios as shown in Figure 8. This evaluation is performed with sysbench and eight guest VMs. The performance of HVD is better than cowloop-SSD where the write operations are more than 30.

4 Conclusion and Future Work

This paper presents a hybrid virtual disk that makes possible the efficient combination of SSD and HDD within consolidated environments. We derive the performance benefit from fast random reads of SSD by locating a read-only template disk image in SSD, while written data are stored in HDD. This placement policy intensifies the advantages of SSD, avoiding overheads caused by write operations. The contribution of this work is that the hybrid CoW storage is obviously advantageous, in terms of performance and cost, for server consolidation workloads, especially for those in which sequential operations might be broken into small random ones.

As future work, we plan to implement sophisticated migration techniques between SSD and HDD. We expect that the identification of write-once read-many data is a crucial concern for the migration work. In addition, we also consider using SSD as a cache that temporarily stores the written blocks from guest VMs. There are lots of related work including the five-minute rule [21], and we will evaluate them in the virtualization environment and HVD. Efficient migration will make the hybrid virtual disk approach more successful for virtualized environments.

References

1. http://www.vmware.com
2. Rhea, S., Cox, R., Pesterev, A.: Fast, Inexpensive Content-Addressed Storage in Foundation. In: Proceedings of the 2008 USENIX Annual Technical Conference (2008)

3. Liguori, A., Van Hensbergen, E.: Experiences with Content Addressable Storage and Virtual Disks. In: Proceedings of the Workshop on I/O Virtualization, WIOV '08 (2008)
4. McLoughlin, M.: The QCOW image format, `http://www.gnome.org/~markmc/qcow-image-format.html`
5. Kotsovinos, E., Moreton, T., Pratt, I., Ross, R., Fraser, K., Hand, S., Harris, T.: Global-scale service deployment in the XenoServer platform. In: Proceedings of the First Workshop on Real, Large Distributed Systems (WORLDS '04), San Francisco, California (December 2004)
6. Meyer, D.T., Aggarwal, G., Cully, B., Lefebvre, G., Feeley, M.J., Hutchinson, N.C., Warfield, A.: Parallax: virtual disks for virtual machines. In: Proceedings of Eurosys (2008)
7. Douglis, F., Caceres, R., Kaashoek, F., Li, K., Marsh, B., Tauber, J.A.: Storage alternatives for mobile computers. In: Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI), November 1994, pp. 25–37 (1994)
8. Park, C., Seo, J., Seo, D., Kim, S., Kim, B.: Cost-efficient memory architecture design of nand flash memory embedded systems. In: Proceedings of the 21st International Conference on Computer Design (ICCD'03), October 2003, pp. 474–480 (2003)
9. Caulfield, A.M., Grupp, L.M., Swanson, S.: Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications. In: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (2009)
10. Gupta, A., Kim, Y., Urgaonkar, B.: DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (2009)
11. Kang, J., Jo, H., Kim, J., Lee, J.: A Superblock-based Flash Translation Layer for NAND Flash Memory. In: Proceedings of the International Conference on Embedded Software (EMSOFT), October 2006, pp. 161–170 (2006) ISBN 1-59593-542-8
12. VMware, Inc. VMware VMFS product datasheet. `http://www.vmware.com/pdf/vmfs_datasheet.pdf`
13. McLoughlin, M.: The QCOW image format, `http://www.gnome.org/~markmc/qcow-image-format.html`
14. Warfield, A., Ross, R., Fraser, K., Limpach, C., Hand, S.: Parallax: Managing storage for a million machines. In: Proceedings of 10th Hot Topics in Operating Systems (May 2005)
15. `http://www.atcomputing.nl/Tools/cowloop/`
16. `http://sysbench.sourceforge.net/`
17. `http://www.mysql.com/`
18. `http://www.seagate.com/staticfiles/docs/pdf/marketing/po_barracuda_7200_12.pdf`
19. `http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=161&partnum=MCCOE64G5MPP#component01`
20. 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 19th ACM Symposium on Operating Systems Principles, pp. 164–177 (2003)
21. Graefe, G.: The Five-Minute Rule 20 Years Later. Communications of the ACM 52(7), 48–59

An Efficient Implementation of GPU Virtualization in High Performance Clusters

José Duato[1], Francisco D. Igual[2], Rafael Mayo[2], Antonio J. Peña[1], Enrique S. Quintana-Ortí[2], and Federico Silla[1]

[1] Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia (UPV), 46022–Valencia, Spain
{jduato,fsilla}@disca.upv.es, apenya@gap.upv.es

[2] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I (UJI), 12071–Castellón, Spain
{figual,mayo,quintana}@icc.uji.es

Abstract. Current high performance clusters are equipped with high bandwidth/low latency networks, lots of processors and nodes, very fast storage systems, etc. However, due to economical and/or power related constraints, in general it is not feasible to provide an accelerating co-processor –such as a graphics processor (GPU)– per node. To overcome this, in this paper we present a GPU virtualization middleware, which makes remote CUDA-compatible GPUs available to all the cluster nodes. The software is implemented on top of the sockets application programming interface, ensuring portability over commodity networks, but it can also be easily adapted to high performance networks.

Keywords: Graphics processors (GPUs), virtualization, high performance computing, clusters, Grid.

1 Introduction

Virtualization of hardware resources is receiving considerable attention in the last years as a means to reduce the economic cost, ease the administration, and provide better security in large data centers [4].

On the other hand, graphics processors are increasingly being adopted as a hardware solution to accelerate computationally-intensive applications [1,13,14]. Improvements in the programmability of these architectures [12,2] and their excellent performance-power ratio will probably generalize their use in large clusters for high performance computing (HPC) in the near future. However, adding one hardware accelerator to every node in an HPC cluster is not efficient, neither from the performance point of view nor from the power consumption perspective, because, on one hand, not all applications can take advantage of the accelerator and therefore there is no need for a large number of them and, on the other hand, current GPUs have a great impact on the overall power consumption of the system[1]. Economic cost, maintenance, and space also advise

[1] A GPU may well increase the power consumption of an HPC node by 20-30%.

against the one GPU per node solution. Thus, future HPC clusters may well include a few of these accelerators in certain nodes of the system.

In this paper we present a prototype middleware that virtualizes a hardware resource like a GPU in an HPC cluster, as a front-end virtualization. This type of virtualization can be implemented by *device emulation*, that is, by providing a complete replication of the entire hardware accelerator, so that the architecture can be emulated on a different one. However, for computationally-intensive applications this approach is not valid due to the emulation overhead. A better choice to service HPC applications is to offer a virtualized hardware platform, time-sharing the real resource among the users. This is the approach we adopt in our proposal by *Application Programming Interface* (API) *remoting*. Therefore, there is no need of hardware support nor silicon changes.

Our middleware offers the possibility of running different parts of an application on different accelerators, dynamically selecting the most suitable one. Although we have focused only on GPUs, our software can be extended to other types of accelerators. Thus, the goal is to offer virtualized CUDA-compatible GPU devices that can be used by all nodes in the cluster with low overhead. Although similar approaches have been recently followed in the field of virtual machines and graphics [5], the specifics of our target environment and CUDA led us to adopt a different approach since, among others, we do not have to take care of visual output or suspend and resume functionality. Instead, we have to deal with CUDA specifics such as streams and execution control.

The rest of this paper is organized as follows: Section 2 presents the details of the proposed virtualization solution. Section 3 introduces performance related issues. In Section 4 we discuss the current development status of our implementation. Next, Section 5 presents some performance results and, finally, Section 6 summarizes the conclusions of our work.

2 Virtualized GPU Architecture

GPUs are integrated devices in the form of cards that are attached to a server with a general-purpose processor via a PCI-Express (PCIe) bus. To exploit the GPU computing power, part of the program has to be written as a *kernel*, which at runtime is sent and executed on the GPU. The GPU driver is in charge of transferring the program, initiating its execution, and handling its completion.

Both the general-purpose server and the GPU feature separate memory maps. Transferring the data required by the kernel and later retrieving back the results is explicitly addressed by the user. Therefore, in an HPC system where a node without a local GPU is running an application that invokes a GPU kernel, we have to provide support for transferring the kernel and data, and dealing with the initiation/completion of the kernel execution on a remote GPU. In particular, our virtualization middleware consists of two parts: the client middleware is installed as a shared library on all nodes of the HPC cluster which have no local GPU; and the server middleware is executed in the node(s) equipped with GPU(s). We name these nodes hereafter as clients and server(s), respectively.

2.1 Implementation

The current implementation of the virtualization software targets the NVIDIA CUDA programming environment and the NVIDIA G80 and GT200 series.

CUDA enables general purpose computing on the latest NVIDIA GPUs in a C-like programming language, exposing the device architecture as a set of SIMD multiprocessors. This smooths the learning curve for the non-expert programmers on graphics-specific programming languages such as OpenGL and Cg. More detailed information about CUDA can be found in [12].

Client. These nodes employ a library of wrappers to the CUDA Runtime API. During the compilation of the application, two different object files are generated exploiting the compiler driver options: the GPU-module comprises the device code image to be executed on the remote GPU; and the CPU-module contains the code that has to be executed on the local general-purpose processor. The global executable file includes all the functionalities required to identify and connect to the server, locate and send the GPU image file, submit requests for the execution of a specific kernel and transfer dynamic data, and receive and pass back to the application the output resulting from the remote execution.

Server. On the server we add a GPU network service which listens for requests on a TCP port. To deal with the low-level GPU-module operations which are not supported by the CUDA Runtime API, this daemon has been implemented using the low-level Driver API. A library scheme of both client and server applications is shown in Figure 1.

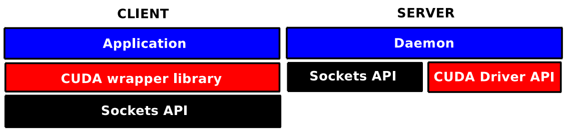


Fig. 1. Scheme for the client and server applications library

The daemon serves requests of CUDA calls on the local GPU. Each remote execution is served by a new process on an independent GPU context. The use of threads for this purpose is not considered an option as potential segmentation faults on Driver API calls could lead to server termination.

In general, the execution of a kernel requires several phases: in the initialization stage, the server receives the GPU code image with the kernels to be executed and the definition of the variables statically allocated by the client application. Once this initialization is completed, the server is able to process a request for executing a kernel. If there is additional data to be used, it must be transferred from client to server before the execution starts. Once the kernel execution is completed, the output data is available to be sent back to the client.

Communication protocol. The data protocol for the communication between client and server has been designed to be as simple as possible, so that communications involve little computation and make an efficient use of network resources. Both data and control flows make use of the network.

Until automatic server discovery is implemented, the first action on the client side is an explicit call to an initialization function, which connects to a specific server. This function automatically locates and sends the application-associated GPU-image file to the server.

In the communication protocol, the first 32 bits of the stream request identify the function which has been called, while the subsequent data is function-dependent, specifying the particular parameters of each function call. The server always sends a 32-bit result code, and possibly more data depending on the requested function.

A sample sequence diagram of the communications generated by a matrix-matrix multiplication execution is shown in Figure 2, which illustrates the following steps:

1. The client application opens a socket connection to the server, where a daemon process is listening. The client then locates and sends the GPU-image to the server, which loads it into a new GPU context. Upon completion, the server sends the result code of the module load operation back.
2. The client requests `memory allocation` on the GPU memory map for the three matrices involved in matrix multiplication. For each one of the three requests, the server replies with the result code of the allocation operation, followed by the pointer to the allocated memory.
3. The next step consists in sending the source data matrices to the GPU memory. To do so, the client sends two `memory copy` requests, each of them specifying the destination pointer, size of the data to be transferred and direction of the copy (from host to device[2]), followed by the corresponding data. Once a request is received and executed, the server sends back the result code of the operation.
4. The GPU is then ready to execute the matrix-matrix multiplication kernel. Next, the client application sends a `launch` request, specifying the kernel to be executed and its execution stack, which consists in a `grid` and `block` configuration, as well as the parameters of the kernel (those commonly required by the BLAS `sgemm` subroutine). Once the launch is done, the server daemon sends back the corresponding result code.
5. At this point, the result of the matrix multiplication is stored in the GPU memory. To transfer it to the local memory, the client application sends a new `memory copy` request, this time specifying the direction as "device to host". The server response is the corresponding result code followed by the requested data (only if the copy operation was successful).
6. Allocated memory is released next. To accomplish this, the client application sends a `free` request per matrix, receiving a result code per request.

[2] In CUDA terminology, `host` stands for a computer holding a CUDA-compatible GPU card, and `device` stands for the GPU itself.

7. The last step consists in calling a `destroy` function, which closes the socket. Upon reception, the daemon server process quits servicing the current execution and releases the associated resources.

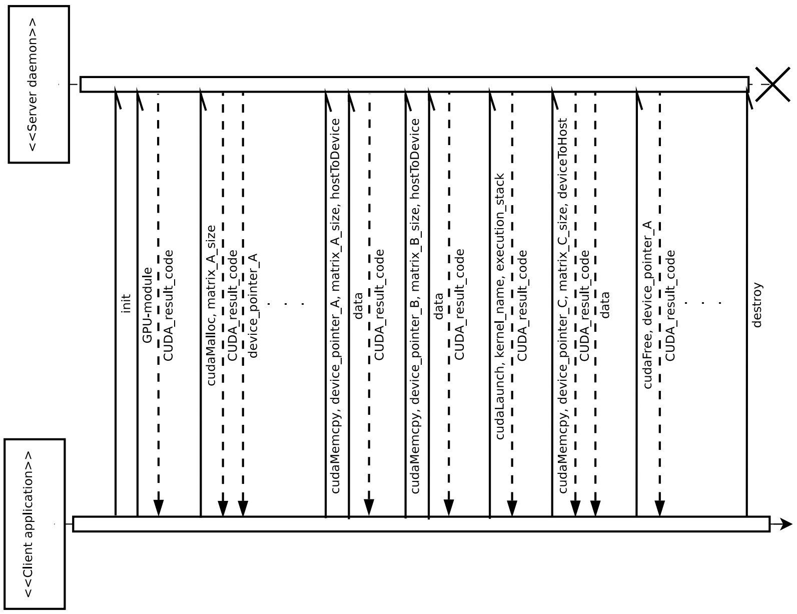

<<Server daemon>>
<<Client application>>
init
GPU-module
CUDA_result_code
cudaMalloc, matrix_A_size
CUDA_result_code
device_pointer_A
. . .
cudaMemcpy, device_pointer_A, matrix_A_size, hostToDevice
data
CUDA_result_code
cudaMemcpy, device_pointer_B, matrix_B_size, hostToDevice
data
CUDA_result_code
cudaLaunch, kernel_name, execution_stack
CUDA_result_code
cudaMemcpy, device_pointer_C, matrix_C_size, deviceToHost
CUDA_result_code
data
cudaFree, device_pointer_A
CUDA_result_code
. . .
destroy

Fig. 2. Matrix-matrix multiplication. Sequence diagram of client-server communications. Memory allocation and release operations are summarized for legibility purposes.

3 Performance Considerations

Virtualizing the GPUs implies an overhead due to the communications over the network, which depends on the specifics of the interconnect (latency and bandwidth). In our approach we intend to explore both the Gigabit Ethernet standard accessed via the sockets API and the new extensions to the HyperTransport (HT) technology, recently proposed in the High Node Count HyperTransport Specification [3]. This technology provides a non-coherent shared memory map for a large number of computing nodes with very low latency, as data transfers are managed by hardware with no intervention from the operating system kernel.

One advantage of the socket-based implementation is that it could be used even in low performance networks for academic purposes, thus offering access to a few high performance GPUs concurrently to all the students. On the other hand, the HT based implementation is expected to offer clients of an HPC cluster seamlessly access to a remote GPU with negligible overhead.

Users of the virtualization middleware must take into account that the usage of asynchronous CUDA calls will notably increase the performance of their applications reducing the communication overhead.

To reduce the response time, our server uses a `prefork` technique as follows:

1. The parent server is started.
2. The parent server creates a child server which will serve all requests from a single remote execution.
3. The child server receives a connection request.
4. The child server communicates this event to its parent.
5. The parent server spawns another child to attend eventual requests.
6. Children terminate after the connection is closed by their respective client.

In addition, another tweak is introduced to save time: before a child server blocks waiting for an upcoming connection request, it pre-initializes the CUDA driver API environment and creates a new context, so it is ready to load a GPU image immediately when it is received.

Finally, to attain high-performance data transmission over a TCP/IP network, Nagles's algorithm [9,10] –TCP layer default congestion control algorithm– has been disabled on both client and server sides. Basically, this algorithm delays the effective sending of TCP frames until a buffer is filled in or a timer expires. This behavior provides good performance in many environments, preventing the transmission of a large number of small packets, which would waste most of the network bandwidth transmitting packet headers. However, in HPC a precise control of the moment a frame must be sent out is desired. In Linux operating system (OS), this is achieved by explicitly choosing the time when the TCP transmission buffer must be flushed by managing TCP layer socket options and policies. A more detailed discussion about Nagle's algorithm can be found in [7].

4 Development Status

This project is, at the moment of the writing, ongoing so this could be considered a proof of concept.

We concentrate our development efforts on the TCP based approach, but we expect to adapt the developed middleware to the HT based interconnect by just changing the communication routines to send and receive data over the network.

4.1 Implemented Functionality

Thus far we have successfully implemented on both client and server sides the following of the CUDA Runtime API:

– Device Management Runtime.
– Thread Management Runtime.
– Event Management Runtime.
– Execution Control Runtime.
– Part of the Memory Management Runtime.
– Error Handling Runtime.

This subset of the API is sufficient to build a series of commonly used applications and obtain some timing results, which are presented in Section 5.

4.2 Future Work

We are working on the completion of the whole CUDA Runtime API. In particular, current missing functionalities comprise:

- Stream Management Runtime.
- Texture Reference Management Runtime.
- Part of the Memory Management Runtime, mainly asynchronous operations.

One drawback of the current implementation is that it needs to keep the CPU and GPU codes in separated files. The GPU code is compiled with the NVIDIA compiler driver `nvcc` using its "device code repositories" feature (see [11]) to obtain the GPU-image file. On the other hand, the CPU code is compiled using a C or C++ compliant compiler (such as GNU or Intel C Compilers) to obtain the final executable. This leads to the unavailability of the `CUDA C language extensions` (such as the simplified kernel call syntax) on host code. This separation is mandatory because during compilation of mixed GPU and CPU code, `nvcc` automatically inserts calls to undocumented CUDA Runtime API library functions, (presumably to allow the application locate the embedded GPU code in the executable, among others). To address this, we will develop a preprocessor which will transparently take care of code separation and compilation.

Another limitation of the mandatory code separation step is the impossibility of using prebuilt CUDA libraries such as CUBLAS, because at the moment our runtime is unable to locate embedded GPU code. However, we expect to overcome this problem because GPU code is easily recognized in the executable.

On the final stage of the development, we will deal with multi-server related functionalities, such as automatic discovery and load balancing.

Additionally, we will explore more network related tweaks, such as TCP `defer accept` and `quick ack` options, and we will adapt our communication routines to the high performance HT based network.

When completed, we may consider adapting the implementation to Windows OS based systems, and the recently emerged OpenCL framework [8].

Finally, in the long term we intend to generalize our implementation to different kinds of accelerators.

5 Results

In this section we evaluate the impact of the virtualization overhead, using two case studies: the product of two matrices and the Fast Fourier Transform (FFT).

The nodes of the cluster employed in the evaluation are equipped with two Quad Core Intel® Xeon® E5410 processors (2.33 GHz, 8 GB RAM), running the Linux OS (kernel 2.6.18). The node interconnect is a Gigabit Ethernet. The GPU is an NVIDIA Tesla C1060 (driver version 180.22) attached to a PCIe 2.0 x16 port[3]. The server daemon has been built over CUDA Toolkit 2.1.

[3] A PCIe 2.0 x16 graphics link features a maximum bandwidth of 8 Gbytes/s.

The matrix-matrix product implemented in routine `sgemm` as part of Intel MKL (v10.1) is employed on the CPU. On the GPU, we have used Volkov's implementation of the matrix-matrix product routine [15], as this is currently the base for the tuned implementation in CUBLAS 2.1. On the other hand, we have used the FFTW library (v3.2) on the CPU and Volkov's FFT implementation on the GPU, over 1024 complex single precision points. To exploit the GPU massive parallel capabilities, our tests comprise different number of FFT operations, provided that GPU is able to compute multiple FFTs in parallel. To accommodate network variability times are averaged over 30 executions.

The left-hand plot in Figure 3 shows that the execution of Volkov's kernel on a virtualized GPU over small and moderate-size matrices is slightly slower than the local CPU implementation in MKL –maximum of 2.5 secs. on a $8,192 \times 8,192$ matrix–, while for large matrices it is up to a 15% faster, saving 10.6 secs. The figure also shows that most of the time is spent in memory transfers, due to network bandwidth limitations. The right-hand plot in the same figure reports that the execution times for the FFT are between 150 and 1,150 msecs. faster in the local CPU than in the remote GPU, once more due to the network limitations.

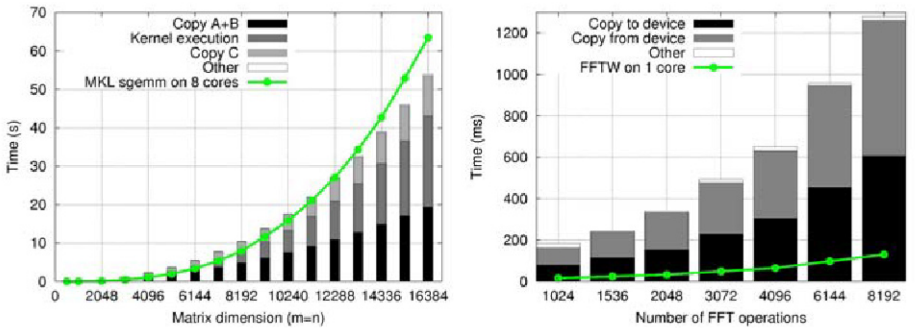


Fig. 3. Left: `sgemm` processing times on CPU vs. virtualized GPU; "Other" includes initialization, memory allocation, memory release and destruction operations. **Right:** `FFT` processing times; "Other" also includes kernel execution.

This result demonstrates that computing a matrix multiplication over a remote GPU can be faster than doing it over the local CPU, even when the connection happens to be a commodity network. However, when the problem requires less computation per data, as is the case for the FFT (the cost of the matrix multiplication is $O(n^3)$ while that of the FFT is $O(n \log n)$, where n is the problem size), there is a serious bottleneck in low bandwidth networks.

A comparison of the execution times of the CUDA function call of our implementation with those of a "local" CUDA execution, reveals that all the former functions are slightly slower (around 10 ms. per call), except for memory copies when a large amount of data is sent over the network (see the plots in Figure 4). In particular, remote copies are around 15 times slower than local ones, yielding a maximum overhead of 28 seconds for the largest problem on `sgemm`.

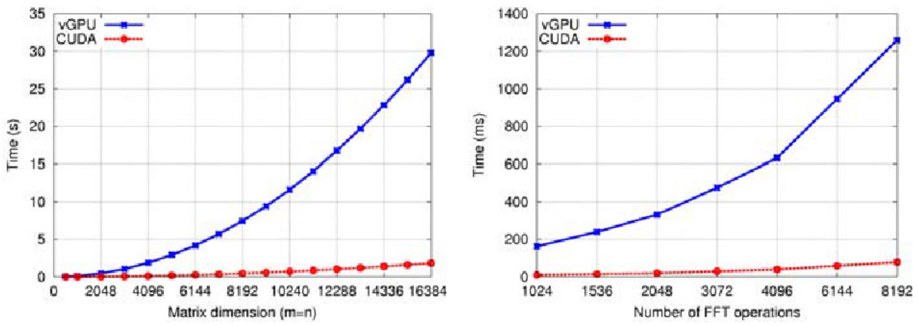


Fig. 4. Left: Execution time of the three matrix copies involved in the matrix-matrix multiplication. **Right:** Time for the memory copy operations (both directions) of FFT.

Those results illustrate that the overhead introduced by our implementation is mostly caused by network related delays, as PCIe bandwidth is an order of magnitude faster than the Gigabit Ethernet one. Therefore, we expect to reach a performance that is close to that obtained with a local GPU execution when the target network is based on HT, as this network will attain 3.2 Gbytes/s, which is –according to our tests– around a half of the effective peak bandwidth of the GPU reads through the PCIe bus. Furthermore, the latency estimations for the HT-based network are around a few μsecs. [6], which is an order of magnitude lower than the latency for a TCP frame on the network used in our tests.

6 Conclusions

We have implemented a GPU virtualization prototype which enables seamlessly remote CUDA Runtime API calls. The middleware enables an efficient use of an HPC cluster where only some of the nodes are equipped with accelerators.

We have shown that our approach can deliver reasonable performance for clusters connected via a commodity network. As a major part of the time is spent on communications, we expect a negligible degradation in performance in case the nodes are connected via a high performance network.

There is much future work in completing the whole CUDA API and solving multi-server related issues. Eventually, we also expect to generalize this solution to OpenCL compatible accelerators.

Acknowledgements

Researchers at UPV were supported by PROMETEO from Generalitat Valenciana under Grant PROMETEO/2008/060.

Researchers at UJI were supported by the Spanish Ministry of Science and FEDER (contract no. TIN2008-06570-C04-01), and by the Fundación Caixa-Castelló/Bancaixa (contract no. P1B-2007-19).

References

1. Barrachina, S., Castillo, M., Igual, F.D., Mayo, R., Quintana-Ortí, E.S.: Solving dense linear systems on graphics processors. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 739–748. Springer, Heidelberg (2008)
2. Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., Hanrahan, P.: Brook for GPUs: stream computing on graphics hardware. In: SIGGRAPH '04: ACM SIGGRAPH 2004 Papers, pp. 777–786. ACM, New York (2004)
3. Duato, J., Silla, F., Yalamanchili, S., Holden, B., Miranda, P., Underhill, J., Cavalli, M., Brüning, U.: Extending HyperTransport protocol for improved scalability. In: Proceedings of the First International Workshop on HyperTransport Research and Applications (WHTRA 2009), pp. 46–53 (2009)
4. Figueiredo, R., Dinda, P.A., Fortes, J.: Guest editors' introduction: Resource virtualization renaissance. Computer 38(5), 28–31 (2005)
5. Andres Lagar-Cavilla, H., Tolia, N., Satyanarayanan, M., de Lara, E.: VMM-independent graphics acceleration. In: VEE '07: Proceedings of the 3rd international conference on Virtual execution environments, pp. 33–43. ACM, New York (2007)
6. Litz, H., Froening, H., Nuessle, M., Bruening, U.: VELO: A novel communication engine for ultra-low latency message transfers. In: ICPP '08. 37th International Conference on Parallel Processing, September 2008, pp. 238–245 (2008)
7. Mogul, J.C., Minshall, G.: Rethinking the TCP nagle algorithm. Computer Communication Review 31(1), 6–20 (2001)
8. Munshi, A. (ed.): OpenCL 1.0 Specification. Khronos OpenCL Working Group (2009)
9. Nagle, J.: Congestion control in IP/TCP internetworks. Computer Communication Review 14(4), 11–17 (1984)
10. Nagle, J.: RFC 896: Congestion control in IP/TCP internetworks (January 1984)
11. NVIDIA: Nvidia CUDA Compiler Driver NVCC. NVIDIA (2008)
12. NVIDIA: Nvidia CUDA Programming Guide Version 2.1. NVIDIA (2008)
13. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26(1), 80–113 (2007)
14. Stone, S.S., Haldar, J.P., Tsao, S.C., Hwu, W.-m.W., Liang, Z.-P., Sutton, B.P.: Accelerating advanced MRI reconstructions on GPUs. In: CF '08: Proceedings of the 2008 conference on Computing frontiers, pp. 261–272. ACM, New York (2008)
15. Volkov, V., Demmel, J.W.: Benchmarking GPUs to tune dense linear algebra. In: SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, Piscataway, NJ, USA, pp. 1–11. IEEE Press, Los Alamitos (2008)

MyriXen: Message Passing in Xen Virtual Machines over Myrinet and Ethernet

Anastassios Nanos and Nectarios Koziris

National Technical University of Athens,
School of Electrical and Computer Engineering,
Computing Systems Laboratory,
Zografou Campus, Zografou 15780, Greece
{ananos,nkoziris}@cslab.ece.ntua.gr

Abstract. Data access in HPC infrastructures is realized via user-level networking and OS-bypass techniques through which nodes can communicate with high bandwidth and low-latency. Virtualizing physical components requires hardware-aided software hypervisors to control I/O device access. As a result, line-rate bandwidth or lower latency message exchange over 10GbE interconnects hosted in Cloud Computing infrastructures can only be achieved by alleviating software overheads imposed by the Virtualization abstraction layers, namely the VMM and the driver domains which hold direct access to I/O devices.

In this paper, we present MyriXen, a framework in which Virtual Machines efficiently share network I/O devices bypassing overheads imposed by the VMM or the driver domains. MyriXen permits VMs to optimally exchange messages with the network via a high performance NIC, leaving security and isolation issues to the Virtualization layers. Smart Myri-10G NICs provide hardware abstractions that facilitate the integration of the MX semantics in the Xen split driver model. With MyriXen, multiple VMs exchange messages using the MX message passing protocol over Myri-10G interfaces as if the NIC was assigned solely to them. We believe that MyriXen can integrate message passing based applications in clusters of VMs provided by Cloud Computing infrastructures with near-native performance.

Keywords: Virtualization, Xen, Myrinet, Ethernet, MyriXen, Myri-10g, Linux, I/O, DMA, Virtualized I/O, Message Passing, MX.

1 Introduction

Current Cloud Computing research is focused on providing a scalable, on-demand, clustered computing environment. One of the major challenges in this field is bridging the gap between Virtualization techniques and high performance network I/O retrieval techniques [1]. To meet the I/O needs of HPC applications running in Virtualization environments, research has focused on alleviating overheads that arise due to intermediate software layers. Hardware vendors [2] have become increasingly aware of this issue and provide the community with smart I/O devices that can export multiple interface instances using software [3] or hardware [4,5].

Integrating I/O Virtualization semantics in HPC infrastructures can both facilitate research and offer software management freedom without affecting isolated application execution. Having a cluster of VMs that is almost identical to a cluster of workstations and can be set up in hours or minutes seems quite intriguing.

While HPC interconnects provide abstractions that can be exploited in Virtual Machine execution environments, they lack architectural support. In this paper, we describe MyriXen, a framework in which Virtual Machines share network I/O devices efficiently bypassing overheads imposed by the VMM or the driver domain model. Specifically, MyriXen allows VMs to optimally exchange messages with the network via a high performance NIC leaving only security and isolation issues to be handled by the hypervisor and the NIC itself. In the following sections, we present some background information followed by MyriXen's design architecture.

2 Background

In Virtualization environments, the basic building blocks of the system (i.e. CPUs, memory and I/O devices) are multiplexed by the Virtual Machine Monitor (VMM). The latter may allow VMs to access these resources directly, in order to maximize performance. Xen [6] consists of the hypervisor, the driver domains and the VMs (guest domains). Driver domains are privileged guests that access I/O devices directly and provide the VMs abstractions to interface with the hardware via a *split driver model*. Driver domains host a *backend* driver while VM kernels host a *frontend* driver exposing a generic device API to guest kernels or userspace. The frontend communicates with the backend via an event channel communication mechanism along with interrupt routing, page–flipping, and shared memory techniques.

In Xen, memory is virtualized in order to provide physically contiguous regions to Operating Systems running on guest domains. This is achieved by adding a a per-domain memory abstraction called *pseudo-physical* memory. Therefore, in Xen, *machine memory* refers to the physical memory of the entire system, whereas pseudo-physical memory refers to the memory regions exported to the Operating Systems running in each guest domain.

Xen Paravirtualized Network I/O. Xen's paravirtualized (PV) network architecture is based on a split driver model. Guest VMs host the *netfront* driver which exports a generic Ethernet API to kernelspace. The driver domain, which directly accesses network hardware, hosts the hardware specific driver, the protocol interface driver, and the *netback* driver. The latter communicates with netfront via a dedicated event channel. Upon initialization of a guest domain, the netfront binds to the netback driver, which in turn binds to a dummy network interface bridged with the physical network interface in software. Thus, network packets originating from the VM are transfered (copied or flipped) to the netback driver and are injected to the NIC via the software bridge.

Contrary to the previous approach, a Xen guest domain can directly access an I/O device at the expense of system's security. For example, suppose a VM accesses directly a generic Ethernet NIC. To achieve maximum bandwidth the VM kernel allocates memory for building an Ethernet frame and, with the help of the VMM, informs the NIC's DMA engine about the physical address of the buffer. The NIC then DMAs data from the VM's space to its packet buffers and emits the frame to the network. However, the DMA transfer begins without checking the validity of the source or destination, which evidently raises security issues.

Xen Grant Mechanism. To efficiently share memory pages across guest domains, Xen exports a *grant* mechanism to guest domains. Xen's grants are stored in *grant tables* and provide a generic mechanism to memory sharing between domains. Network I/O device drivers are based on this mechanism in order to exchange control information and data via shared memory. Each domain has its own grant table.

Myrinet/MX basics. In order to fully comprehend MyriXen's design, it is of utmost importance to present the basic structure of Myrinet/Myrinet eXpress (MX) [7]. To run MX in Virtualization environments, these features have to be integrated into Xen's device driver architecture.

Myrinet [8] is a low-latency, high bandwidth interconnection infrastructure for clusters. Two generations of Myrinet are currently available: Myrinet-2000 and Myri-10G. Myrinet achieves low-latency cut-through switching using source routing. Myri-10G is based on the same physical layer as 10GbE and can either use the source-routed Myrinet protocol or 10GbE as the Data Link layer.

Myrinet NICs feature a RISC microprocessor called Lanai, which consists of: the CPU, the copy engine, two packet interfaces, each with its own on-chip send and receive packet buffers, a Z-port (XAUI Myri-10G / 10G Ethernet), and a PCI Express port. The Local Bus (LBUS) is an interface to a fast, synchronous, static SRAM.

To reduce the overhead of OS involvement, Myrinet employs user-level networking techniques. In this model, an application process is allowed to control the Network Interface (NI) directly; since the OS is no longer invoked for communication, its role is undertaken by a combination of application-level libraries and firmware executing on the NIC, while data exchange between the two is set up by privileged code inside an OS kernel module.

To provide user-level networking facilities to applications, the MX message passing system is used. An application is granted control of the NI by mapping part of the NI memory space into its own virtual memory. User-level communication is accomplished by using unprivileged load/store instructions to the relevant VM segments bypassing OS abstractions and copies. This is a privileged operation, which is done via system calls to the MX kernel module during the application's initialization phase. Each of these parts, called MX endpoints, acts as an isolated virtual network interface at the process level for the application and contains an unprotected part, which is mapped to userspace, and a protected,

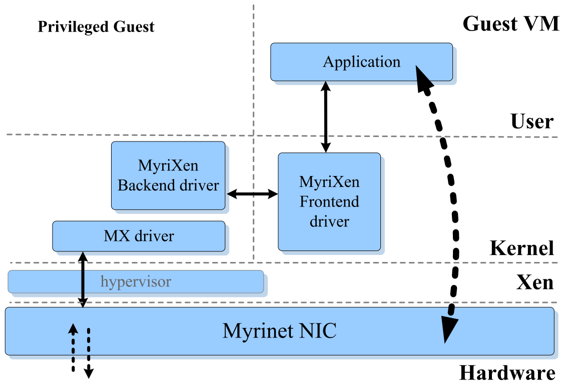


Fig. 1. MyriXen

trusted part, which is only accessible by the kernel module and the firmware. An endpoint provides an entry point to the interconnect's hardware, protected from other processes, with fairness relative to the other endpoints opened on the same NIC.

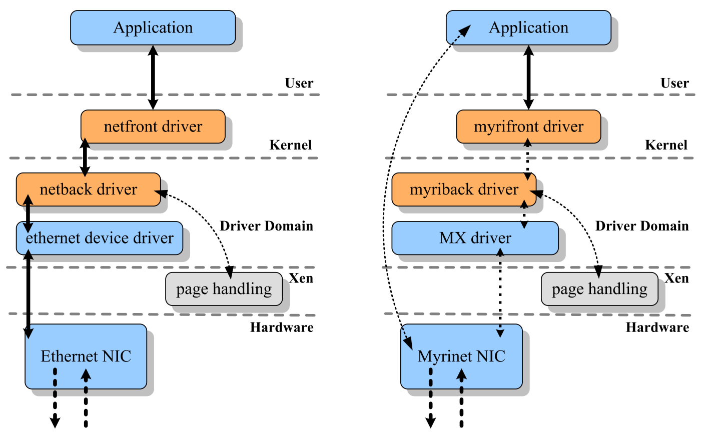
3 MyriXen

Like most device drivers, the MX driver cannot export multiple interfaces for a single Myrinet NIC. Thus, a split driver model approach is required for multiple Xen VMs to share a single NIC. In the following sections we describe our prototype design.

Architecture. Fig 1 illustrates the basic design architecture of MyriXen. The backend runs on top of the native MX driver in the driver domain. It waits for incoming requests from the frontend drivers running in the VMs. The frontend driver replaces the core MX driver and, once loaded, establishes two event channels with the backend driver: the first one is used to process requests initiated from the VM; the second one is used for informing the guest domain about MX events.

The frontend driver is a relatively thin layer and exports to VM userspace the same API as the core MX driver. One of its tasks is the installation of mappings for the MX library to access the NIC directly. Moreover, its role is to provide the mechanism to open and close MX endpoints via the event channel mechanism. The backend driver sets up the Myrinet NIC to support these features and issues acknowledgments for event completion or error.

The split driver model used in Xen poses difficulties for user-level direct NIC access in Xen VMs. To enable driver domain bypass techniques, we need to let VMs have direct access to certain NIC resources. The building block of MyriXen is *myriback* which allows *myrifront* to communicate with the MX core driver, and thus, install the prerequisites to send or receive a message to / from the network.



(a) Xen PV software stack (b) MyriXen software stack

Fig. 2. Xen and MyriXen network I/O Software Stack

The myrifront driver, similarly to the netfront driver, communicates with the backend via an event channel mechanism. Contrary to the netfront / netback architecture, MyriXen utilizes the backend in conjunction with the core MX driver to grant pages to the VM user space and install mappings that can simulate the normal case; the netfront driver, on the other hand, uses these channels to send or receive packets (as a data path). Fig. 2(a) shows Xen PV net I/O software stack as opposed to MyriXen's software stack in Fig. 2(b).

MyriXen Semantics. To communicate with the network, an application running in a VM needs privileged access to the NIC. In order to provide isolated, virtualized access to the Myri-10G NIC we must take into account the following issues:

Initialization: For applications to communicate using MX, the MX library along with the NIC have to be initialized. The preparation steps needed for initialization include allocating basic structures for the library to communicate with the Lanai. This is essentially a mapping operation: the VMs forward requests to the myriback driver and wait for completion; the backend driver executes the operation and provides the acknowledgment back to the frontend. The frontend manages these resources using handles that are provided by the backend via the event channel mechanism.

Endpoint management: Message passing over the network occurs between endpoints. Thus, a VM has to open and close an endpoint before and after communication takes place, respectively. Opening an MX endpoint means obtaining a specific handle from the Lanai and the MX core driver. This is realized in two steps. During the first step, the frontend requests an endpoint. The backend gets informed via the relevant event channel and requests a free endpoint from the NIC. The NIC returns a handle to that endpoint. The second step consists of the acknowledgment sent back to the frontend along with the handle.

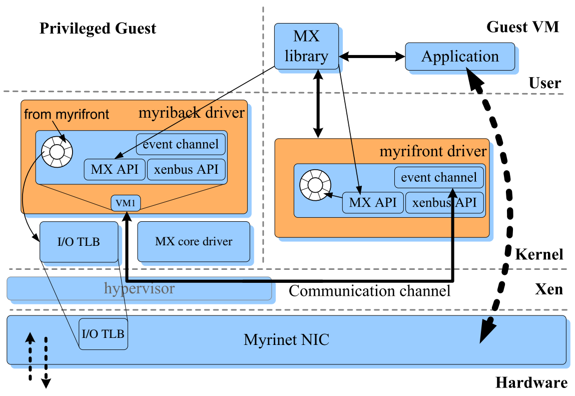


Fig. 3. MyriXen network I/O architecture

Memory registration: After the Initialization and the Endpoint opening phase, the frontend has to register memory regions that will participate in the message exchange. The memory registration requests originate from the frontend and provide grant references to the Xen hypervisor. The MX core driver registers these memory regions to the Lanai, and forms an on-chip page table cache. This is a necessary step in order to realize zero-copy data transfer.

Message Matching: MX offers a rich message matching interface to applications and application libraries in order to alleviate matching overhead imposed by the kernel or generally OS involvement. Thus, MyriXen can exploit the existing MX semantics for matching to achieve direct delivery of messages to VMs bypassing receive-path copies that impose significant overhead.

Protection: As mentioned in section 2, Xen uses a grant mechanism to transfer pages from privileged to guest domains and vice-versa. MX is based on zero-copy and OS-bypass techniques to achieve line-rate bandwidth for 10Gbps networks. However, these techniques can lead to significant throughput degradation due to Xen's architecture. To avoid illegitimate access to arbitrary memory regions, an I/O TLB cache mechanism can be used to transfer valid page tables to the Lanai SRAM. Thus, memory protection is guarantied by having the Lanai check for valid memory regions before programming its DMA engines.

Address Translation: DMA transfers between userspace buffers and Lanai's on-chip packet buffers are invoked by applications that have already opened an endpoint and have registered specific memory regions. The registration process involves pinning pages that will be used for DMA transfers as well as informing the MX core driver and the NIC about these pages. Address translation in Xen consists of two steps: the first is a virt-to-phys translation on the VM side (myrifront) and the second is a pseudo-phys-to-machine translation done be the hypervisor so as to end up with addresses that the Lanai DMA engines are able to follow.

Discussion. Note that initialization, endpoint management, and memory registration are steps that occur outside the critical path of network intensive applications. Although these operations are resource intensive (PIO to the NIC, asynchronous events, grant references, etc.), they take place before or after the communication phase, and thus, their performance impact is not significant.

In MyriXen, data flows directly from applications to the NIC leaving only control issues to be handled by the hypervisor or the driver domain. Applications control the DMA engines of the Lanai chip directly using load/store instructions to NIC memory mapped regions.

An important feature of our design is the decoupling of data transfers from the Virtualization layers. The implications of this mechanism for the overall throughput constitute a possible caveat of our approach. Specifically, the way the control path interferes with data communication may result in significant overhead, which needs to be examined with extensive performance evaluation using microbenchmarks.

Finally, one could also criticize how MyriXen handles isolation: MX provides isolated access by using MX endpoints exporting a virtual NI to any application that requests access to the network. Our design of MyriXen is such that MX semantics to applications remain unaltered, so the isolation features provided by MX would also characterize MyriXen.

4 Related Work

Previous work has concluded that the integration of Virtualization semantics in specialized software running on Network Processors can isolate and finally minimize the hypervisor and driver domain overheard associated with device access. Liu et al. [9] describe VMM-bypass I/O using the Infiniband architecture. Their approach is novel and based on Xen's split driver model. Many features presented in this work can be used by modern Virtualization platforms to overcome the bandwidth and latency limitations imposed by software overheads. Although their model is thoroughly designed, they focus on Infiniband, which uses specialized hardware without open specifications.

Mansley et al. [10] developed a safe, direct data path between the VM and the network using Solarflare Ethernet NICs. However, their approach is device specific and cannot be used for efficient message passing, since existing messaging protocols have to be stacked above Ethernet. Our approach accounts for this problem and aims to keep MX's full compatibility with the applications.

Santos et al. [1] present a performance analysis of network device I/O in Xen and identify possible bottlenecks. They also propose optimizations and implement a small subset of them. The primary objective of this work is to present a generic framework for sharing smart NICs in Xen VMs. As an extension to [1], in [3], the authors describe mechanisms to overcome the bandwidth limitations imposed by the VMM and the driver domain model in Xen and provide an efficient and direct data path for VMs to access the network. Unfortunately, the authors do not focus on HPC environments although the features they propose could be exploited by message passing protocols.

While the aforementioned studies present a promising framework they have only been implemented on top of Ethernet ignoring the potential advantages of Myrinet. This study aspires to provide insight into integrating existing messaging protocol semantics to Virtualization platforms.

5 Conclusions and Future Work

We have described the basic design of MyriXen, a thin split driver layer on top of the Myri-10G MX driver to support message passing in Xen VMs over the wire protocols supported in Myri-10G infrastructures. Message passing occurs in a direct I/O data path leaving only control and management issues to the driver domains and the VMM. MyriXen's design is based on the Xen split driver model in order to sustain manageable infrastructures that can provide VMs with security, isolation, and migration capabilities even in heterogeneous hardware configurations.

MPI applications over a cluster of VMs are liable to significant performance degradation due to limited network performance [11]. This is mainly caused by overheads imposed by the driver domains, which directly access I/O devices. MyriXen accounts for this problem by combining the following two important features: it utilizes the Xen split driver model and, at the same time, installs a direct application-to-NIC data path for message exchange.

We believe this approach is a step towards integrating HPC applications in Virtualization environments, such as Cloud Computing infrastructures. MyriXen provides VM-level networking semantics in an already deployed Virtualization platform, Xen, with a vast user base. Our future work will be firmly oriented towards evaluating our prototype design and presenting an extensive performance evaluation of MyriXen in conjunction with latency breakdown analysis on MPI applications running on clusters of VMs. We believe that there is scope for these applications to benefit greatly from MyriXen's direct data path and achieve performance close to native.

References

1. Santos, J.R., Turner, Y., Janakiraman, G., Pratt, I.A.: Bridging the gap between software and hardware techniques for I/O Virtualization. In: ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference, pp. 29–42. USENIX Association, Berkeley (2008)
2. Abramson, D., Jackson, J., Muthrasanallur, S., Neiger, G., Regnier, G., Sankaran, R., Schoinas, I., Uhlig, R., Vembu, B., Wiegert, J.: Intel Virtualization Technology for Directed I/O. Intel Technology Journal 10(03), 179–192 (2006)
3. Ram, K.K., Santos, J.R., Turner, Y., Cox, A.L., Rixner, S.: Achieving 10 Gb/s using safe and transparent network interface virtualization. In: VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pp. 61–70. ACM, New York (2009)

4. Raj, H., Schwan, K.: High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. In: HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing, pp. 179–188. ACM, New York (2007)
5. LeVasseur, J., Panayappan, R., Skoglund, E., du Toit, C., Lynch, L., Ward, A., Rao, D., Neugebauer, R., McAuley, D.: Standardized But Flexible I/O for Self-Virtualizing Devices. In: Ben-Yehuda, M., Cox, A.L., Rixner, S. (eds.) Workshop on I/O Virtualization. USENIX Association (2008)
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I.A., Warfield, A.: Xen and the Art of Virtualization. In: SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164–177. ACM, New York (2003)
7. Myricom: Myrinet eXpress (MX): A High Performance, Low-Level, Message-Passing Interface for Myrinet (2006), `http://www.myri.com/scs/MX/doc/mx.pdf`
8. Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., Su, W.: Myrinet: A Gigabit-per-Second Local Area Network. IEEE Micro 15(1), 29–36 (1995)
9. Liu, J., Huang, W., Abali, B., Panda, K.: High performance VMM-bypass I/O in virtual machines. In: ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference, p. 3. USENIX Association, Berkeley (2006)
10. Mansley, K., Law, G., Riddoch, D.: Getting 10 Gb/s from Xen: Safe and Fast Device Access from Unprivileged Domains. In: Euro-Par 2007 Workshops: Parallel Processing (2007)
11. Youseff, L., Wolski, R., Gorda, B., Krintz, C.: Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems. In: First International Workshop on Virtualization Technology in Distributed Computing, VTDC 2006 (2006)

Virtage: Server Virtualization with Hardware Transparency

Hitoshi Ueno, Satomi Hasegawa, and Tomohide Hasegawa

Enterprise Server Division, Hitachi, Ltd.
Hadano, Japan
{hitoshi.ueno.uv,satomi.hasegawa.ch,
tomohide.hasegawa.tv}@hitachi.com

Abstract. This paper presents Virtage, Hitachi's virtualization technology, which enables logical partitioning of server platforms. Logical partitioning architecture brings two benefits to this virtualization technology, one is hardware transparency and another is better performance.

We first describe its key feature: hardware transparency. With the advantage of hardware transparency for logical servers, it is possible to provide the same guest Operating System (OS) interface from both physical servers (non-virtualized servers) and logical servers. Virtage is hypervisor-type virtualization, and therefore has a natural performance advantage over host-emulation virtualization offerings because guest OSs can be simply and directly executed on the virtualized environment without host intervention.

Then we demonstrate this lower overhead for CPU and I/O with performance experiments and explain how Virtage brings mainframe-class virtualization to blade servers. In this paper we study the factors of virtualization overhead for CPU and I/O by using original event monitoring tool that can get hypervisor-event information.

Keywords: virtualization technology; Virtage; server; pass-through method; hardware transparency; benchmark; scalability.

1 Introduction

Recent developments in server hardware technology have resulted in increased numbers of CPU cores and the need to install greater memory capacity on each server, thereby making it harder to efficiently utilize the full performance of a server with a single Operating System (OS). This problem can be resolved through server virtualization technology, which allows one server to run multiple applications on top of independent OSs. Currently, server virtualization technology is mainly used for server migration from old and low performance systems to new high performance hardware. It is important to have high performance, high availability and easy system management for server consolidation.

Virtage is the new server virtualization technology available on Hitachi's BladeSymphony servers. Virtage implements a logical partitioning system, which presents the physical server hardware configuration to logical servers without abstraction of the hardware configuration. This is a unique virtualization feature. By using this logical partitioning system, Virtage users can utilize high availability cluster systems or database systems.

The purpose of this paper is to introduce Virtage and this new approach of server virtualization technology. It is suitable for a wide range of business and research applications. We discuss the design aspects of Virtage and its advantages. In the rest of the paper, we describe the key features of Virtage with some examples of system implementation and operations management. We also present some experimental results of Virtage performance.

2 Design Policy

For blade servers—which are often employed in businesses' mission-critical systems—running a virtualized environment, it is desirable to maintain support for high-availability and high-operability systems in order to secure the same high reliability and operational efficiency as physical servers.

Research was carried out in the field of mainframe computing systems in the 1970s in pursuit of higher performance in virtual machines [2], [3]. Even today, products based on logical partitioning are heavily used in mainframe computing systems.

Our development goal is a server product intended for mission-critical systems, which requires a virtualization system suitable for this kind of usage profile. For this reason, Virtage adopts the logical partitioning method.

Virtage has been designed to provide "hardware transparency" for logical servers. It provides the same guest OS interface for both logical servers and physical servers. The logical partitioning of systems provides the following benefits:

- The same file system can be used in both the virtual and the physical server environment. (e.g. NTFS, EXT3, etc.) With this Virtage feature, a system disk which was deployed in a virtual environment can also be used in a physical environment.
- Both a CPU dedicated mode and a CPU shared mode are available. The CPU service ratio can be assigned in units of one percent when using CPU shared mode.
- Less performance overhead caused by the virtualization control.
- In a logical server environment, I/O intensive application software works normally, just as it works on a physical server. Examples are storage management software and cluster control software of hot-standby systems.

In the logical server environment of Virtage, any application software can work normally, because a guest OS and its applications can access I/O devices in the same way they would access the physical server's hardware interface.

2.1 Hardware Transparency

In this section, we discuss hardware transparency, which is a distinctive characteristic of Virtage. Here, hardware transparency is defined according to the two conditions specified below. We will examine what benefit is delivered by hardware transparency when these two conditions are satisfied.

1. All I/O commands issued by the guest OSs (OSs on virtual computers) and their responses must be identical to those on a physical server.
2. The format of the disk used by the guest OSs must be identical to that of a physical server.

If the process executed by a guest OS on the disk is just a simple read/write, condition 1 does not necessarily need to be satisfied. In general, neither condition 1 nor condition 2 is satisfied with other virtualization software. However, in order to return correct responses to the execution of control-related commands issued by cluster control software etc., condition 1 should be satisfied. To put it another way, when this condition is met, a cluster of linked logical servers can be structured without special restrictions.

Meanwhile, two advantages are obtained when condition 2 is met. One advantage is that, because the file system can be accessed directly, the system can be accessed from a backup server that does not support virtualization. This makes it possible to adopt the so-called LAN-free backup configuration, in which the incremental change data of the disk used by the guest OSs is backed up, file by file, via SAN, without going over the LAN.

The other advantage arises from the ability to write data from the guest OSs to the disk unit securely. In general, virtualization software often adopts a virtualized file system as the disk unit's recording format. In this case, it is common to prepare a disk cache in the server's memory. If the virtualization software has a disk cache, data is likely to be retained in the cache for a certain period of time after a data-write-completion-response has been returned, even when the guest OSs execute a raw write (intending to record data securely on a physical disk). Such caching means that the system cannot maintain the journal files necessary to preserve the reliability of transaction monitors and database software. Some virtualization software does have raw write capability: a guest OS can write data directly into disk units, bypassing the virtualized file system. However, to use this capability, users have to consider which volumes must be configured with virtualized file systems and which volumes must be configured with normal file systems. This adds to the complexity of configuring production systems.

2.2 Comparison of I/O Virtualization Methods

There are two typical I/O virtualization methods. One is the pass-through method and the other is the hypervisor emulation method, which is often used in other virtualization software. The pass-through method is desirable in order to achieve hardware transparency.

- Pass-through method

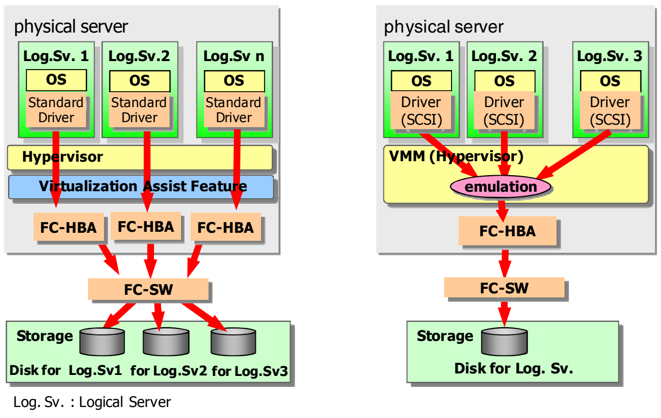
  The pass-through method is a technique that does not need the intervention of the hypervisor when a guest OS activates and executes an I/O operation.

  The DMA address set up by the guest OS on the I/O device is a virtualized memory address. Therefore, if the I/O device executes data transfer as is, the data will be transferred to the wrong memory location. To prevent this error without the intervention of the hypervisor, hardware assistance is required.

- Hypervisor emulation method

  The hypervisor emulation method is a virtualization technique that executes address translation for DMA transfer by guest OSs by means of emulation, using hypervisor traps at the time of I/O activation.

Fig. 1 shows conceptual diagrams of the two virtualization methods, and Table 1 compares their functional features.



(a) Pass-through method (b) Hypervisor emulation method

Fig. 1. Structure of I/O virtualization methods

Table 1. Comparison of I/O virtualization methods

	Pass-through Method		Hypervisor Emulation Method	
I/O performance	G	High performance by direct execution	N	Large overhead due to emulation
Hardware transparency	G	Clustering for hot standby system is available	N	Hard to gain right response for control accesses
File system transparency	G	LAN free backup/ DBMS use available	N	Virtual file system prevents common system use
Disk virtualization	N	not supported	G	Virtual Machine migration available

G: Good, N: Not good

On the basis of the comparison in Table 1, we decided to adopt the pass-through method for virtualization on the BladeSymphony BS1000 servers, which is aimed at businesses' mission-critical systems.

As discussed above, hardware assistance is needed to implement pass-through I/O. We developed a unique I/O virtualization assist mechanism to implement this hardware assistance. Thus, we have succeeded in creating a practical pass-through method.

3 Performance Experiments

In a real-life virtualized environment, some virtualization control overhead cannot be ignored, such as emulation processing. We have made a performance experiments with Virtage. It is a system-level test with a 3-tier application program running on several logical servers (LPARs) on a physical server in order to investigate the scalability of this workload.

In this test, we used a typical ERP workload model, Sell from Stock load Scenario, for the system test because it is CPU and memory intensive and uses substantial amounts of network traffic and disk I/O.

Fig. 2 shows the test environment used. The system used for the experiments consisted of application servers on logical servers (one application server per LPAR), a database server on a separate physical server, and external test drivers connected to the application servers.

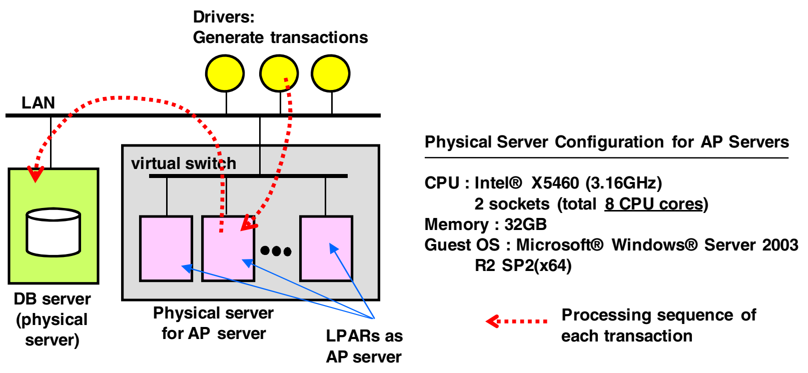

Drivers:
Generate transactions
LAN
virtual switch
DB server (physical server)
Physical server for AP server
LPARs as AP server
Physical Server Configuration for AP Servers
CPU : Intel® X5460 (3.16GHz)
2 sockets (total 8 CPU cores)
Memory : 32GB
Guest OS : Microsoft® Windows® Server 2003 R2 SP2(x64)
Processing sequence of each transaction

Fig. 2. 3-tier (database server, application servers, and the drivers) application program model and the environment

We evaluated three levels of transaction workloads; heavy, middle, and light with 60%, 40%, and 20% of CPU utilization in the system respectively. The performance is given by measuring the response time of the transactions with these workloads. The test server has eight physical CPU cores. In the test, we used four, six, and eight logical servers (LPARs) as application servers, and each application server had two virtual CPUs. Therefore, a maximum of 16 virtual CPUs is running in this test.

In the case of four running application servers, the number of virtual CPU cores equals the number of physical CPU cores. However, with six and eight application servers, the number of virtual CPU cores exceeds the number of physical CPU cores. We call this processor over-commitment. This ERP workload test is designed to be completed with a two second response time for practical use.

Fig. 3 shows that this condition can be satisfied in the cases of four and six logical servers (LPARs) running simultaneously in one physical server for all levels of transactions. On the other hand, in the test with eight logical servers, this condition is satisfied only for the middle and light transaction workloads.

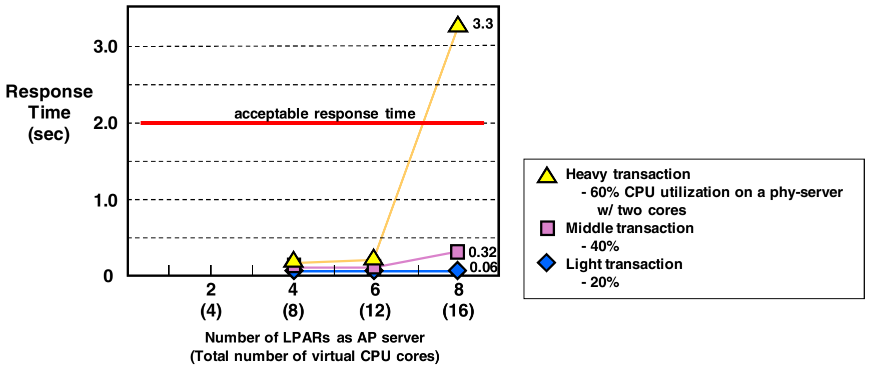


Fig. 3. Virtage performance of ERP workload

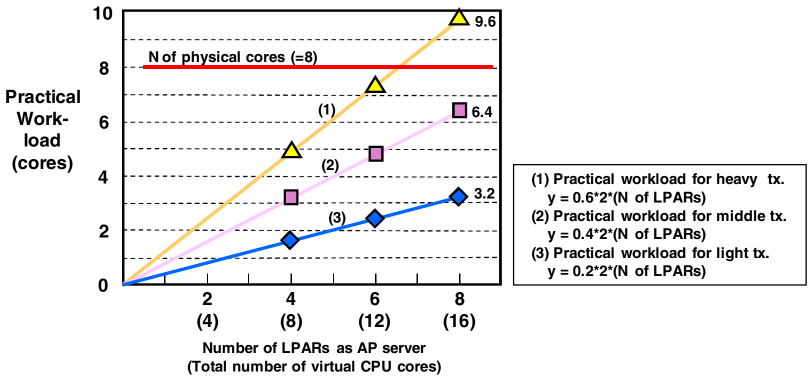


Fig. 4. Practical workload of each transaction case

We can easily understand above phenomenon if we have the graph which plots practical consumed physical CPU cores. Fig. 4 shows numbers of CPU cores that consumed in each case, heavy, middle and light. It shows that only one case exceeds numbers of existing "physical CPU cores" that is 8 and Fig. 3 shows that only this case exceeds the response time limitation. In other words, if numbers of consumed CPU core is less than numbers of existing physical CPU cores, the system achieves enough quick response time.

Therefore, relatively good performance with processor over-commitment is shown for light and middle transaction workloads. Only the heavy transaction workload with eight logical servers cannot achieve the desired performance. We are investigating whether this is due to a performance limitation of this test model (that is, an application limitation) or whether it is due to virtualization overhead.

4 Performance Overhead Analisys

"CPU performance overhead" in server virtualization is the additional processing time which does not exist on physical server processing. From viewpoints of instruction execution time, some hypervisor instruction streams are inserted in the application and OS instruction stream in virtualization environment. The performance overhead is the additional time to execute the inserted hypervisor instruction streams.

Examples of causes of the inserted hypervisor instruction streams are as follows.

– Critical resource accessing : If guest OSs access critical resources like control registers or I/O registers for MMIO operations, the hypervisor has to emulate the resources.
– Page fault occurrence : If page fault is occurred on guest OSs, the hypervisor has to maintain memory management tables like page table or TLB.
– Interrupt from hardware : If external interruptions are occurred on the hardware, hypervisor has to emulate another interruption for specified guest OS.

It is useful to measure the frequency and execution time to detect the cause of performance overhead and think out the measures, because the frequency of those hypervisor events depends on the characteristic of the executions of application programs.

We developed the event counter and the event tracer functions as a performance monitor and put them into the hypervisor. The event counter counts the hypervisor events and sum up their execution time respectively. The event tracer records the hypervisor events in sequence.

We measured the virtualization overhead factors using Virtage performance monitor in former ERP workload. The Intel® X5460 platform, Virtage uses SPT(Shadow Page Tables) method for guest to host address translation. The overhead is relatively high because the hypervisor execution is needed for address translation. Concretely hypervisor set the guest to host translation to SPT at Page fault and invalidate it at CR access which is issue by OS at guest process switch.

The Intel® X5570 platform, Virtage uses EPT(Extended Page Tables) method for guest to host address translation. In EPT method the address translation is executed by hardware and hypervisor execution is not needed. The overhead is 76.6% less than the one in SPT method.

Table 2. Result of performance monitoring

Factor of VMexit	Grouping	SPT method	EPT method
PF_REASON_INST_EMULATION	MMIO	0.053	0.000
mmioCacheEmulation	MMIO	0.000	0.000
External interrupt	Interrupt	0.033	0.054
Interrupt window	Interrupt	0.003	0.001
CPUID	Instruction Emulation	0.000	0.000
MOV DR	Instruction Emulation	0.000	0.000
I/O instruction	ACPI timer	0.002	0.002
RDMSR	Instruction Emulation	0.000	0.000
WRMSR	Instruction Emulation	0.000	0.000
TPR below threshold	Interrupt	0.003	0.003
APIC(EOI)	Interrupt	0.017	0.033
APIC(other)	Interrupt	0.090	0.088
vmexitEptViol_COST	MMIO	0.000	0.052
HLT(host)	Halt	0.009	0.001
PF_REASON_SPT_UPDATE	Page Fault	0.482	0.000
CR access	CR access	0.306	0.000
INVLPG	MMU	0.001	0.000
PF_REASON_GUEST_PF	MMU	0.001	0.000
Total		1.0	0.234

Table 2 shows the result of the performance monitoring for ERP workload benchmark, and Fig. 5 shows performance overhead analysis results between SPT method and EPT method in the workload case.

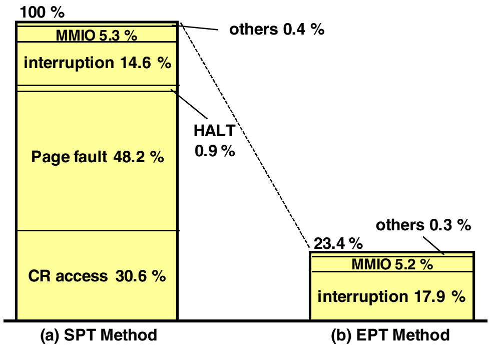


Fig. 5. Performance overhead analysis

It shows that the principal cause of overhead by SPT method is Page Fault and CR access. Page Fault occurs when the address required by running program is not mapped to the physical address. In virtualization environment, a hypervisor translates the address that required by guest program to the physical address on the host side and registers the translation information into SPT when Page Fault occurs. As long as the translation information exists in SPT, when the same page is accessed, processor refers SPT and converts guest address to host physical address. Therefore, Page Fault does not occur. The overhead depends on how often new pages access occur. In this workload, new page access occurred with certain frequency. Consequently, it can be seen a lot of Page Fault overhead. CR access occurs when OS switch the executing process. With the occurrence of CR access, Hypervisor switch SPT. This cause another overhead on CR access if there are many processes switches.

Here we compare it with former research for such analysis of virtualization overhead.

Apparao[13] points out that processing cost increase 600% for context switching when numbers of VM increase from 1 to 4, in their virtualization evaluation environment using Xen hypervisor and SPECjbb2005 workload. It also points out that reasons of the large processing cost are TLB flush, page walk and cache pollution.

Our workload and virtualization software environment is different to above research, but our measurement results takes on similar aspect to it from viewpoint that the primary factor of overhead is management of context switch.

The analysis of here is from a viewpoint of hypervisor software processing, and former research analysis is based on hardware processing factor. That is, the cause of "page fault" and "CR access" in fig. 5 is context switch. Hypervisor makes another SPT for new context, and it changes current page table contents for new SPT, and after that TLB is flushed. Along with execution of new context, page walk process is configured by hardware.

Page fault overhead time measured in fig. 5 includes these particular actions on hypervisor. Similarly CR access on guest OS needs particular actions made by hypervisor. Guest OS accesses logical CR to dispatch new process, hypervisor traps it and emulates for physical CR access with validity check.

EPT method has a feature that processor is able to translate from guest address to host address directly. It is not necessary SPT management. With no hypervisor intervention, no overhead is achieved with EPT method. As a result, if there is a lot of SPT update such as this workload, substantial performance improvement can be expected with applying EPT method.

Fig. 5(b) shows overhead factors of same ERP benchmark workload when the system uses Intel® X5570 processor. Factor of page fault and CR access are suppressed in EPT method, those are appeared in SPT method, and only the overhead which is related to I/O operations like MMIO or interruptions are remain.

At the result, in principle 78.8% of overhead in SPT method is suppressed because of supporting EPT hardware (Fig.5(a)), in practice the overhead of EPT method is smaller 76.6% than SPT method (Fig.5(b)).

This case must be treated carefully as one case, it is not applicable for all cases, but we can understand that EPT method is extremely good control method for achieving low performance overhead.

5 Concluding Remarks

This paper describes a new virtualization technology and presents a comparison between implementations of hardware transparency by software virtualization and hardware-assisted virtualization. In addition, with some performance experiments, we show that Hitachi's virtualization technology, Virtage, has a wide range of practical applications, from the enterprise business area to the high performance computing area.

For achieving its performance, Virtage has performance monitor functions, and it helps resolving much performance problems on virtualization environment.

Hardware transparency on Virtage simplifies complicated system configurations for enterprise systems thanks to the hardware assist features for I/O control. This cannot be achieved with pure software virtualization. The hardware transparency not only enables low emulation overhead from the hypervisor, but it also enables efficient virtualization control, high availability, and high operability.

We examined the performance of virtualization overheads and scalability under different workloads within a Virtage environment. This experiment indicates that Virtage has small virtualization overhead and relatively good performance scalability in practical use with processor over-commitment. Virtage is ready to use for a wide range of applications in the business and high performance computing areas.

However, since with Virtage the guest OS can directly access the hardware specifications of the physical server, it does not meet some user requirements. For example, that a new server installed with a new chipset should be able to support the old version of an OS running on old servers. The new server might be considered unsupported hardware for the old OS.

Our future work will be in the areas of enhancing hardware transparency and satisfying user needs for supporting old OSs on new servers with new chipsets. Also, additional performance evaluation of Virtage in more realistic situations is needed.

References

1. Ueno, H., et al.: Virtage: Hitachi's Virtualization Technology. In: GPC 2009 Workshop Proceedings of Conference, pp. 121–127. IEEE-CS, Los Alamitos (2009)
2. Umeno, H., et al.: Development of a High Performance Virtual Machine System and Performance Measurements for it. IPSJ J. Information Processing 4, 68–78 (1981)
3. Goldberg, R.P.: Survey of Virtual Machine Research. IEEE Computer, 33–45 (1974)
4. Smith, J.E., Nair, R.: The Architecture of Virtual Machines. IEEE Computer, 32–38 (2005)
5. Uhlig, R., et al.: Intel Virtualization Technology. IEEE Computer, 48–56 (2005)
6. Smith, J.E., Nair, R.: Virtual Machines, Versatile Platforms for Systems and Processes. Elsevier Inc., Amsterdam (2005)
7. Barham, A., et al.: Xen and the Art of Virtualization. In: Proc. The 19th ACM Symposium on Operating Systems Principles, pp. 164–177 (2003)
8. Umeno, H., Tanaka, S.: New Methods for Realizing Plural Near-Native Performance Virtual Machines. IEEE Transactions on Computers C-36(9), 1076–1087 (1987)
9. Creasy, R.J.: The Origin of the VM/370 Time-Sharing-System. IBM J. Research and Development, 483–490 (1981)
10. Popek, G.J., Kline, C.S.: The PDP-11 virtual machine architecture: A case study. ACM SIGOPS Operating System Review 9(5), 97–105 (1975)
11. Adair, R., et al.: A Virtual Machine System for the 360/40: Cambridge Scientific Center Report No. G320-2007 (1966)
12. Gil, N., et al.: Intel® Virtualization Technology: Hardware support for efficient processor virtualization. Intel® Virtualization Technology 10(03) (2006)
13. Apparao, P., et al.: Architectural Characterization of VM Scaling on an SMP Machine. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 464–473. Springer, Heidelberg (2006)

Scalable Repositories for Virtual Clusters

Paolo Anedda, Simone Leo, Massimo Gaggero, and Gianluigi Zanetti

CRS4 Distributed Computing Group, Edificio 1, Polaris, Pula, Italy
`{firstname.lastname}@crs4.it`,
`http://dc.crs4.it`

Abstract. For a large class of scientific data analysis applications it is becoming important, due to the sheer size of datasets, to have the option to perform the analysis directly where the data are stored, rather than on remote computational clusters. A possible strategy is the use of virtual clusters, thus guaranteeing a high degree of isolation from the underlying physical computational structure, and a very compact initial description. Deploying, saving and restoring HPC dedicated virtual clusters introduces, however, a different class of requirements on the virtual machines managing infrastructure, in particular for what concerns storage I/O requirements, whose scalability boundaries are easily reached. Here we discuss an alternative approach based on a storage model that leverages the WORM (write once, read many) character of the data used by VM management to increase, in a scalable way, the aggregate data bandwidth available to virtual cluster level operations and provide preliminary results indicating that it is a viable solution.

1 Introduction

Current scientific data production technologies allow for the collection of huge datasets at a constantly decreasing price. Examples of research fields recently hit by what has been called the "data deluge" include geosciences with embedded networked sensing [1], life sciences with biomedical imaging [2] and high-throughput DNA sequencing [3]. Developing new technologies capable of properly managing these datasets constitutes a precondition for the efficient extraction of knowledge from the data and an interesting research problem in itself.

Applications for any nontrivial analysis of such datasets are usually parallel, and require large computing clusters to be effective. Due to their high installation and maintenance costs, large clusters are almost always shared between research groups with different, possibly conflicting software requirements, making their administration problematic. Moreover, when datasets reach this size, it is usually much more efficient to analyze them directly where they are stored, rather than moving them to a remote computational cluster.

A popular solution to this class of problems is OS-level virtualization, which allows to encapsulate applications, along with all their requirements down to the operating system, into virtual machines (VMs) thus guaranteeing a high degree of isolation and easy migration between different physical hosts. In particular, the paravirtualization approach, adopted by leading technologies like Xen [4,5], allows VMs to achieve extremely low performance overhead,[1] which makes them particularly attractive for running HPC applications. Since scientific data are typically analyzed by means of parallel applications, virtualization leads to computational entities which take the form of *virtual clusters* (VCs) [6], sets of VMs which are deployed and managed as single, self-consistent atomic entities.

A VM for data-driven applications like the ones cited above often needs to be described by a state (file system image and allocated memory) measuring several gigabytes. For example, a recently developed MapReduce [7] application [8] for the analysis of deep sequencing datasets employs VMs with 4 GB of RAM and more than 3 GB of disk space for local caching.

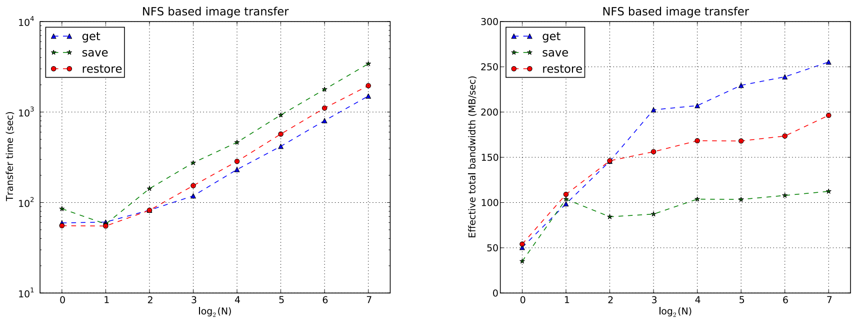


Fig. 1. Left: direct measurements of the total time taken to, respectively, *get* from a central NFS repository the same 3 GB image to N VM host nodes, *save* back to the repository N 3 GB images taken from the same N VM hosts and *restore* the N images from the repository to N VM hosts. Right: corresponding effective bandwidth defined as the total amount of data transferred divided by the total time taken by the transfer. The dashed lines are drawn only to guide the eye. The measurements were obtained using the setup described in section 4.1.

In this context, a virtual cluster's initial description is usually given in terms of N initially identical virtual machines, all of which are instantiated from the same disk image; subsequent events in a typical VC lifecycle include non-destructive save/restore and shutdown. Thus, while the initial cluster deployment involves transferring the same image to N VM hosts, other operations require the saving and, at restart, the restoring, of N different images from/to N different VM hosts. Fig. 1, on the left, shows direct measurements of the total time taken to, respectively, *get* from a central NFS repository the same 3 GB image to N VM host nodes, *save* back to the repository N 3 GB images taken from the same N VM hosts and *restore* the N images from the repository to N VM hosts. The right side of the same figure shows the corresponding effective bandwidth, defined as the total amount of data transferred divided by the total time taken

[1] www.xen.org/about/paravirtualization.html

by the transfer. The measurements were obtained using the setup described in section 4.1. All these processes are parallel, in the sense that all transfers to and from the VM hosts are started synchronously.

It is apparent from figure 1 that, while NFS is a perfectly adequate solution for small clusters, it will not scale as the number of nodes increases. Apart from the specifics of a given hardware setup, this is a direct consequence of having an external fixed storage system, whose bandwidth is independent from the computational cluster's size. On the other hand, NFS provides much more (e.g., supports arbitrary modification on files) than the simple streaming I/O required for handling VM images.

In this paper we discuss an alternative storage approach based on HDFS, the Hadoop[2] Distributed File System. Differently from general purpose, POSIX compliant, distributed file systems such as Lustre [9] and GPFS [10], HDFS is specialized to a model where files, once written, cannot be modified. This dramatically simplifies the management of distributed data coherence while guaranteeing high throughput for streaming applications. Although limited in general file system terms, HDFS is perfectly adequate to VC deployment and migration operations since the latter create what are, essentially, immutable data.

The main point here is that whatever the file system used, it should be able to use storage on the physical VM hosting cluster, and guarantee scalability. Our specific choice was HDFS – it could have been a similar storage system such as CloudStore[3] – because our main virtual cluster applications are Hadoop based and expect the hosting facility to export an HDFS file system.

Our preliminary results indicate that this is indeed a viable solution and that, at the cost of a moderate increase in the complexity of the VM hosts setup, it brings good scalability and dramatic performance improvements with respect to traditional, *out of the cluster* storage strategies.

The rest of the paper is organized as follows: section 2 discusses related work; in section 3 we briefly introduce HDFS; section 4 describes experimental setup and test results; finally, in section 5 we present our conclusions and plans for future work.

2 Related Work

Virtual clusters have been the subject of intense research activities in the past years. In [11], OS and network virtualization are used to partition machines into separate "virtual domains" in order to increase isolation between different organizational units and optimize resource utilization. In [6], a VC is defined as an aggregation of atomic virtual workspaces [12], implemented as sets of virtual machines. Maestro-VC [13] provides on-demand virtual clusters, implemented as sets of Xen VMs, which are deployed and managed as homogeneous entities. The problem of efficient and scalable VC installation is addressed in [14] by means

[2] `http://hadoop.apache.org`
[3] `http://kosmosfs.sourceforge.net`

of pipelined data transfer and automatic caching of frequently used VM images. In [15], VC deployment and configuration is discussed in an HPC context.

These works are mainly focused on extracting homogeneous, customized environments from a broader, possibly heterogeneous resource pool by means of automated installation frameworks. Although it shares similar goals, our work specifically focuses on deployment systems capable of supporting large images (in the order of several gigabytes) deployed on hundreds of cluster nodes.

In this work we use Hadoop HDFS as distributed VM image repository. Alternative solutions, often available in HPC cluster installations, include GPFS [10], PVFS [16] and Lustre [9]. However, given the WORM storage model that characterizes VC management, a specialized file system capable of harnessing it to maximize performance was the natural choice. Concurrent to our work, in an effort to support MapReduce on GPFS, IBM researchers have shown [17] how a standard cluster file system, after undergoing substantial modification, could achieve performances comparable to those of more specialized file system like HDFS, albeit with increased network traffic.

3 Technologies

The main technologies used in the work described here are HDFS and NFS. While the latter, being widely adopted, is well known, the former deserves a short introduction which we will present in the next section.

3.1 HDFS

Hadoop[4] is a popular open source implementation of MapReduce [7], a parallel programming framework initially developed by Google. Hadoop includes a distributed file system for application data storage called HDFS (Hadoop Distributed File System). HDFS has been specifically designed for very high scalability (thousands of nodes, hundreds of millions of files, tens of petabytes) and optimized for high throughput processes on very large datasets.

Its key features are:

- fault tolerance: data blocks are replicated according to a configurable replication factor so that the MapReduce engine can reassign failed tasks to other nodes;
- WORM (write once, read many) storage model: once a file is written, it can never be modified. This allows to maximize the aggregate bandwidth without resorting to complex synchronization mechanisms.

HDFS adopts a master/slave architecture: the master, called *namenode*, stores file system metadata and provides a namespace which allows groups of data blocks to be seen as ordinary files; the slaves, called *datanodes*, physically store data blocks and serve read/write requests from clients. By design, user data never flows through the namenode: when a client requests a file, the namenode

[4] `http://hadoop.apache.org`

simply replies with a set of block IDs and the addresses of the datanodes on which those blocks are stored; actual data transfer happens between the client(s) and the datanodes.

The usual HDFS application is to support the MapReduce computational framework, typically deployed over the same worker nodes as HDFS. Specifically, MapReduce, in its Map phase (where all data records are independent) uses information on data block location to optimize bandwidth to computation by scheduling tasks closest (in a network topology sense – e.g., on the same physical nodes or in the same rack) to where blocks are stored and minimize network traffic: by doing this it is possible, in principle, to reach an effective bandwidth that scales with the number of datanodes.

In the context of this paper, however, we are using HDFS in a different way. We are essentially interested only in reading and writing large numbers of VM image files, concurrently from multiple clients. Thus, while in its usual application HDFS clients independently read different data chunks corresponding to the different sections of the MapReduce input stream, in the VM repository case all clients read/write in sequence the data blocks that constitute each file. In this case parallelism is governed by a pipeline where each stage corresponds to a block and uses a group of datanodes to serve the operation. Therefore, even in the hypothetical case of an infinite size cluster, the depth of the pipeline controls the maximum bandwidth that can be achieved in a *get* operation where N clients concurrently download the same image file from HDFS. A simple model yields the following for the total time T needed to download from an HDFS cluster of S nodes to N clients a file of size $W = Bb_s$, where b_s is the HDFS block size and B the number of blocks, in the limit of $S \geq B$ and large N:

$$T = \frac{W}{b_w}\left(n_t + \frac{t_l}{\tau}\right)\left(1 + \frac{N - n_tR}{n_tRB}\right), \tag{1}$$

where b_w is the point-to-point bandwidth between a client and a datanode, n_t the number of server threads per datanode (in Hadoop, n_t defaults to 3), $\tau = b_s/b_w$ is the time needed to read a block of size b_s, t_l the latency involved in starting a block read, which ranges from tens to hundreds of milliseconds [18], while R is the HDFS block replication factor and by large N we mean $N \gg n_tR$. Eq. 1 predicts an essentially N-independent transfer time for $(N - n_tR) \ll n_tRB$. Accordingly, the effective bandwidth $E_{bw} = NW/T$ will initially grow linearly with N and then saturate at about RBb_w. Of course, this should be regarded as an indication of a general trend: the picture is more complicated for N smaller than n_tR and large N behaviour is also controlled by the co-location of server and client processes and network effects. Differently from *get*, the many-to-many operations *save* and *restore* are in principle parallel since they involve N independent pipelines. However, their scalability is limited by finite size effects when the number of client and server datanodes becomes comparable with the hosting cluster size, especially because of competition for disk access.

4 Experimental Results

We have conducted a series of experiments to assess the feasibility of employing HDFS as a VC repository system by running various configurations on a medium size production cluster.

4.1 Setup

Image transport tests were conducted on a cluster of 384 HP BL460c blades equipped with two quad-core Intel E5440 (2.8 GHz) CPUs, 16 GB of RAM, two 250 GB SATA hard disks and two BCM5708S Gigabit Ethernet NICs. The

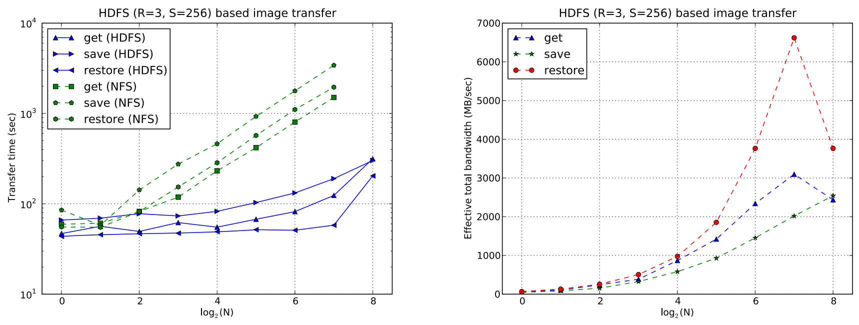


Fig. 2. Left: direct measurements of the total time taken to, respectively, *get* from a distributed HDFS repository the same 3 GB image to N VM host nodes, *save* back to the HDFS repository N 3 GB images taken from the same N VM host nodes, and *restore* the N images back from the repository to N VM hosts. As a matter of comparison, the figure also reports analogous measurements done using the NFS-based repository. HDFS was run with the default settings, i.e., block size equal to 64 MB and replication factor set to three. Right: corresponding effective bandwidth defined as the total amount of data transferred divided by the total time taken by the transfer. The lines are drawn only to guide the eye. The measurements were obtained using the setup described in section 4.1.

computing blades are contained, in groups of 16, in blade enclosures, in turn interconnected by 10GbE links to a central switch so that, albeit with potential latency hits, the cluster can support 1GbE wire speed interconnections between any arbitrary pair of blades.

The cluster is a shared production environment managed with Sun Grid Engine (SGE) [19]: to instantiate HDFS on it, we modified Hadoop On Demand[5] in order to use SGE (instead of its default, TORQUE) as its resource manager.

The external storage system used for NFS tests is a Sun StorageTek 5320 NAS Appliance which mounts 400 GB fiber channel and 1 TB SATA disks for a total capacity of 460 TB.

[5] `http://hadoop.apache.org/core/docs/current/hod_user_guide.html`

4.2 Measurements

In the following, we will use N to indicate the number of VC nodes and S to denote the number of nodes of the hosting physical cluster and thus of the HDFS file system. All scaling measurements are performed on the aforementioned production cluster that did not, at that time, support virtualization. On the other hand, here we are only considering image transport to/from the HDFS repository to the virtualization hosts and thus the reported measures are expected to be relevant to a full VC hosting setup. All measurements are the results of averages on multiple runs and were done while trying to minimize, as much as possible, the impact of external effects such as node sharing with other jobs and conflicts on network resources. The results can, therefore, be considered as best case data that, however, are expected to provide relevant general information on an actual VC hosting production configuration.

Fig. 2 shows, on the left, the results of direct measurements of the total time taken to, respectively, *get* from a distributed HDFS repository the same 3 GB image to N VM host nodes, *save* back to the HDFS repository N 3 GB images taken from the same N VM host nodes, and *restore* the N images back from the repository to N VM hosts. As a matter of comparison, the figure also reports analogous measurements done using the NFS based repository. HDFS was run with the default settings, i.e., block size equal to 64 MB and replication factor set to three. Unless specifically mentioned, we maintained these default settings for all the reported measurements. The simulated physical hosting cluster contained 256 machines, two of which were dedicated to the HDFS and MapReduce masters (respectively the namenode and the job tracker, with the latter unused), while the remaining 254 acted as HDFS slaves (datanodes). HDFS was configured to use only one of the two disks available on each node, while all per-host image read and write operations used the other one. As it can be seen from the figure, the two groups of curves start differing significantly for cluster sizes larger than four. The HDFS repository has a very weakly N-dependent behaviour up to VC clusters of size 32, after which competition between HDFS server and client processes starts to be relevant. It should be noted, however, that HDFS is still at least one order of magnitude better than NFS. On the right side of fig. 2 is shown the corresponding effective total bandwidth, defined as the total amount of data transferred divided by the total time taken by the transfer. As in the previous case, there is a finite size effect for large VC sizes.

Fig. 3 shows how the time taken by an image transfer for a given cluster size N depends on the size S of the physical hosting cluster (and thus of the HDFS repository). On the left is shown the case of the 1-to-N *get* operation. Transfer times seem to rapidly converge to their asymptotic, S-independent, values. Things are qualitatively different, as shown on the right side of fig. 3, for the N-to-N transport operation required by *restore.* Convergence is slower, probably due to competition for chunk reading on the datanodes.

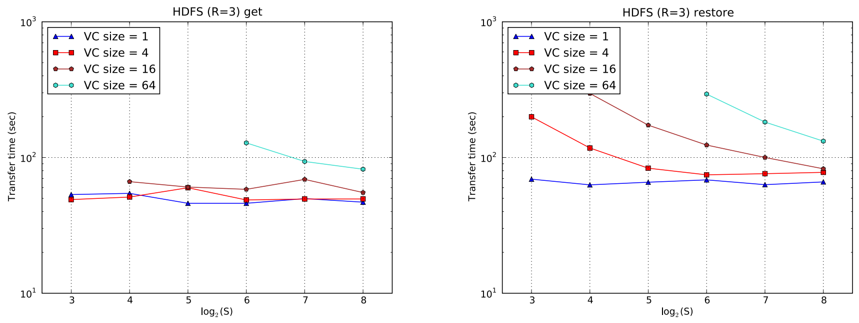


Fig. 3. Dependency of transfer time on the size of the physical hosting cluster S (and thus of the HDFS repository) for different VC size (N) values. Left: the 1-to-N *get* operation. Right: the N-to-N transport operation required by *restore*. Note the difference in convergence to the asymptotic values. The lines are drawn only to guide the eye.

Fig. 4 shows transfer timings – respectively *get* (left), *save* (center) and *restore* (right) – for a fixed physical hosting cluster size $S = 256$, as a function of VC size N for three different replication factors. Note how, as expected, the save operation is the most affected by R, especially as N becomes comparable to S.

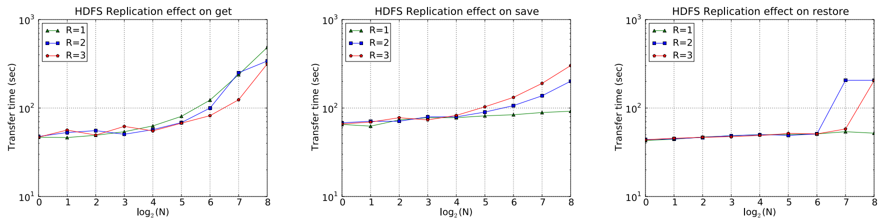


Fig. 4. *get* (left), *save* (center) and *restore* (right) timings for a fixed physical hosting cluster size $S = 256$, as a function of VC size N for three different replication factors. The lines are drawn only to guide the eye.

5 Conclusions and Future Work

We have shown that by using HDFS, a simple specialized distributed file system, it is possible, at the cost of a moderate increase in the complexity of the VM hosts setup, to provide – in a scalable way – the aggregate data bandwidth needed by HPC virtual cluster level management operations.

Measurements were obtained, as much as possible, while trying to minimize the impact of external effects such as node sharing with other jobs and conflicts on network resources. Although the results reported here can, therefore, be considered as best-scenario data, we expect them to provide relevant general information on actual VC hosting production configurations.

Future work will concentrate on analyzing the impact of the proposed solution on production virtual clusters. We also intend to compare HDFS with other distributed file systems (e.g., CloudStore) and to explore issues related to dynamical modifications in size and topology of the hosting physical cluster, both through modeling and experiments.

Acknowledgments. The work described here was partially supported by the Italian Ministry of Research under the CYBERSAR project. We would like to thank L. Leoni, C. Podda, M. Moro and M. Vocale for providing us with precious technical advice. We would also like to thank the anonymous reviewers for helpful comments and suggestions.

References

1. Borgman, C.L., Wallis, J.C., Mayernik, M.S., Pepe, A.: Drowning in data: digital library architecture to support scientific use of embedded sensor networks. In: 7th ACM/IEEE-CS joint conference on Digital libraries (2007)
2. Peng, H.: Bioimage informatics: a new area of engineering biology. Bioinformatics 24(17), 1827–1836 (2008)
3. Editorial: Prepare for the deluge. Nature Biotechnology 26(10), 1099 (2008)
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: 19th ACM Symposium on Operating Systems Principles (2003)
5. Chisnall, D.: The Definitive Guide to the Xen Hypervisor. Prentice-Hall, Englewood Cliffs (2007)
6. Foster, I., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B., Zhang, X.: Virtual clusters for grid communities. In: 6th IEEE International Symposium on Cluster Computing and the Grid (2006)
7. Dean, J., Ghemawat, S.: MapReduce: Simplified DataProcessing on Large Clusters. In: OSDI 2004: Sixth Symposium on Operating System Design and Implementation (2004)
8. Leo, S., Anedda, P., Gaggero, M., Zanetti, G.: Using virtual clusters to decouple computation and data management in high throughput analysis applications
9. Schwan, P.: Lustre: building a file system for 1000-node clusters. In: Proceedings of the 2003 Linux Symposium (2003)
10. Schmuck, F., Haskin, R.: GPFS: a shared-disk file system for large computing clusters. In: Proceedings of the First Conference on File and Storage Technologies (FAST), pp. 231–244 (2002)
11. Ruth, P., McGachey, P., Xu, D.: VioCluster: Virtualization for dynamic computational domains. IEEE International Cluster Computing (2005)
12. Keahey, K., Foster, I., Freeman, T., 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)
13. Kiyanclar, N., Koenig, G., Yurcik, W.: Maestro-VC: A paravirtualized execution environment for secure on-demand cluster computing. In: 6th IEEE International Symposium on Cluster Computing and the Grid Workshops (2006)
14. Nishimura, H., Maruyama, N., Matsuoka, S.: Virtual clusters on the fly – fast, scalable, and flexible installation. In: 7th IEEE International Symposium on Cluster Computing and the Grid (2007)

15. Begnum, K., Disney, M.: Scalable Deployment and Configuration of High-Performance Virtual Clusters. In: 3rd International Conference on Cluster and Grid Computing Systems (2006)
16. Carns, P., Ligon III, W., Ross, R., Thakur, R.: PVFS: a parallel file system for linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference (2000)
17. Ananthanarayanan, R., Gupta, K., Pandey, P., Pucha, H., Sarkar, P., Shah, M., Tewari, R.: Cloud analytics: do we really need to reinvent the storage stack? In: Workshop on Hot Topics in Cloud Computing (HotCloud '09) (2009)
18. Lin, J., Bahety, A., Konda, S., Mahindrakar, S.: Low-latency, high-throughput access to static global resources within the Hadoop framework. Technical Report HCIL-2009-01, University of Maryland, Human-Computer Interaction Lab. (2009)
19. Gentzsch, W.: Sun grid engine: Towards creating a compute power grid. In: First IEEE/ACM International Symposium on Cluster Computing and the Grid (2001)

Dynamic Virtual Cluster Reconfiguration for Efficient IaaS Provisioning

Vittorio Manetti, Pasquale Di Gennaro, Roberto Bifulco, Roberto Canonico, and Giorgio Ventre

University of Napoli Federico II, Italy
Dipartimento di Informatica e Sistemistica
Via Claudio, 21 - 80125 - Napoli, Italy
{name.surname}@unina.it

Abstract. Cloud computing is an emerging paradigm to provide *Infrastructure as a Service* (IaaS). In this paper we present NEPTUNE-IaaS, a software system able to support the whole lifecycle of IaaS provisioning in a Virtual Cluster environment. Our system allows interactive design of complex system topologies and their efficient mapping onto the available physical resources of a cluster. It also provides transparent VM migration features across geographically distributed datacenters, thanks to the adoption of the Service Switching paradigm. We also evaluate the effectiveness of the VM mapping procedures and compare our solution against other existing IaaS solutions.

Keywords: Cloud Computing, IaaS, Xen, Virtual Networking.

1 Introduction

Cloud Computing is an innovative computing model in which "dynamically scalable and often virtualised resources are provided as a service over the Internet" [1]. Following what happened in the last century with electric power or water distribution infrastructures, Cloud Computing enables users to access computing resources on an as-needed basis, relieving them from the responsibility of buying and managing a dedicated computing infrastructure. Cloud providers, on the other hand, can take advantage of scale economies to organize and manage big datacenters, whose ICT resources can be efficiently used by partitioning and renting them to a number of customers. Depending on the abstraction level of the provided resources, Cloud Computing takes different names: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

Originally born as a cluster based network emulation system [2], NEPTUNE-IaaS is a software system developed at University of Napoli Federico II that allows interactive design of networked virtual infrastructures on geographically distributed datacenters, to help provisioning of "Infrastructures as a Service". Our system consists of an interactive client/server software system used to provide users with the possibility of describing and designing the desired virtual infrastructure and of a set of other components that make it possible for services deployed at a given datacenter to be transparently migrated in remote datacenters for load balancing or fault/disaster recovery. NEPTUNE-IaaS is based on the use of Xen for virtualization of computing elements. Xen features are also used to multiplex the communication resources (e.g. network interfaces) available in the cluster nodes among several logically distinct virtualized nodes. Transparent migration of Virtual Machines in NEPTUNE-IaaS is implemented through the adoption of Service Switching, a novel paradigm that aims at extending the concept of virtualization to network services, by decoupling service execution environments and their physical location.

The rest of the paper is organized as follows. In Section II we present NEPTUNE-IaaS, its architecture, the web-based management application we have developed to manage the whole lifecycle of virtual infrastructures. In Section III we present *Service Switching* and its role in our system. In Section IV we present the algorithm we use to efficiently map Virtual Machines onto a cluster's physical resources. Finally, in Section V we briefly compare NEPTUNE-IaaS against two reference IaaS solutions and draw our conclusions.

2 NEPTUNE-IaaS

NEPTUNE-IaaS is a software system for provisioning of IaaS services. In the context of NEPTUNE-IaaS, a *Virtual Infrastructure* is a collection of Virtual Machines provided as a service to an end-user. Virtual Machines are deployed on a subset of a cluster's physical nodes and properly configured according to the user requirements in terms of computational resources, software configuration, virtual network topology, and so on. A Virtual Infrastructure presents at least one public IP address, that is used to make the infrastructure accessible from the public Internet (Entry Point). In general, public IP addresses are assigned only to a subset of the nodes of a Virtual Infrastructure. Other nodes are assigned private IP addresses and can be reached only through the Entry Point nodes. A typical Virtual Infrastructure comprises a NAT/firewall node and a set of backend service nodes, whose NICs are assigned private IP addresses. We will describe later in this paper that the necessity of supporting transparent migration of Virtual Infrastructures across geographically distributed datacenters calls for unique assignment of private IP addresses within a Service Switching domain.

To achieve higher degrees of scalability and resource efficiency, Virtual Infrastructures are instantiated by allocating multiple Virtual Machines onto each of the cluster's real nodes (*node multiplexing*). Likewise, multiple virtual links are multiplexed onto the same shared physical link by associating each virtual link endpoint to a different virtual NIC (*link multiplexing*). Multiple fully isolated Virtual Infrastructures can be concurrently hosted by NEPTUNE-IaaS in the same datacenter, providing users with the illusion of having allocated a dedicated infrastructure.

2.1 NEPTUNE-IaaS Architecture

A cluster managed by NEPTUNE-IaaS (Figure 1) is composed of three components: i) a set of worker nodes providing computational resources used to reproduce emulated networks, ii) a centralized repository providing storage space to worker nodes and iii) a front-end node, *Neptune Manager*. By NEPTUNE-IaaS we intend the whole collection of system software, of which the management software running in the Neptune Manager front-end is the most relevant part. All the physical components of the cluster are connected by two switched LANs, one for "control traffic" (e.g. node configuration) and another for "operational traffic" (i.e. traffic generated by users' applications).

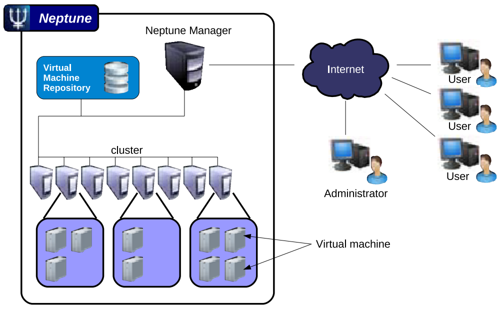


Fig. 1. NEPTUNE architecture

2.2 Virtual Infrastructure Life-Cycle

A Virtual Infrastructure life-cycle can be described by a Finite State Machine (Figure 2). A Virtual Infrastructure life-cycle begins with the definition of a virtual network topology. Once the topology is defined, the infrastructure can be allocated onto the cluster's physical nodes. On user demand, a running Virtual Infrastructure can be either suspended for future reallocation or definitively terminated. Allocation of infrastructures onto the cluster is made under control of system administrators, who need to explicitly accept users requests. Once accepted, an infrastructure's topology allocation process starts. Such allocation process is automatic, involving tasks like virtual nodes mapping on cluster's physical nodes and IP addresses assignments.

To define a Virtual Infrastructure, users can either write a topology description in a custom XML format, defining nodes'properties (NICs, RAM, software configuration, etc.) and links'properties (bandwidth, end points, etc.) or use an interactive graphic tool embedded into the web user interface (Figure 3). The tool assists the description of any node or link property suggesting available choices to the user. It is also possible for users to select pre-defined topologies for fast infrastructure definition. To define virtual nodes software configuration, users can access a "Virtual Nodes Template Images Repository" and select a VM template for each of the virtual nodes. VM templates can be modified and saved as new templates for reuse.

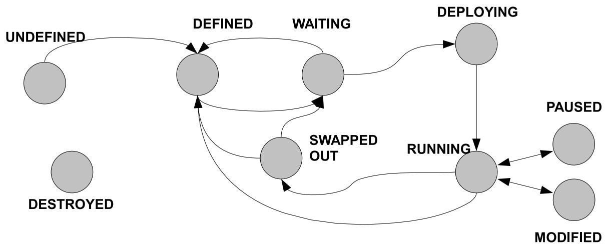
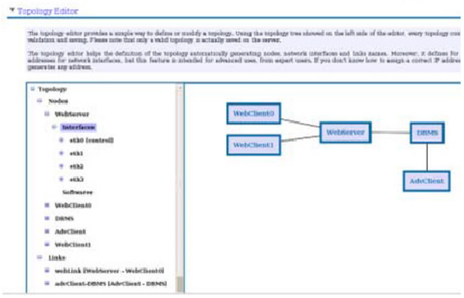


Fig. 2. Virtual Infrastructure lifecycle


Topology Editor

The topology editor provides a simple way to define or modify a topology. Using the topology tree showed on the left side of the editor, every topology con validation and saving. Please note that only a valid topology is actually saved on the server.

The topology editor helps the definition of the topology automatically generating nodes, network interfaces and links names. Moreover, it defines for addresses for network interfaces, but this feature is intended for advanced use, from expert users. If you don't know how to assign a correct IP addres generates any address.

Fig. 3. Interactive editor

2.3 Implementation Details

Node multiplexing is implemented in NEPTUNE-IaaS by means of Xen [3]. Our current implementation relies on the libvirt virtualization API [4], making it feasible supporting different virtualization technologies in the future. The NEPTUNE-IaaS Management Node is responsible of managing Virtual Machines lifecycle.

Mapping of virtual nodes onto the cluster physical nodes is described by an allocation map which can be generated either manually by a system administrator or automatically, by means of a software module implementing a Lin-Kernighan derived optimization algorithm (described in Section 4).

When a virtual network is to be deployed on the physical cluster, Neptune Manager distributes Virtual Machine template instances to the physical cluster nodes. This distribution process is composed of two phases for each virtual node: 1) raw copy of the virtual machine image file containing VM template, and 2) VM creation on the target virtual machine monitor. During this last phase, virtual hardware resources are provided to the virtual node according to node definition provided by the Virtual Infrastructure topology description.

A major problem when dealing with the creation of virtual links is the need to assign IP addresses to both ends of virtual links, according to a general IP addressing scheme. NEPTUNE-IaaS provides an algorithm that automatically assigns subnets to links and IP addresses to their end-points. Furthermore, since several infrastructures can be running on the same shared infrastructure, this algorithm also ensures non overlapping of address spaces used by different infrastructures.

3 The Service Switching Paradigm

Service mobility is a key feature for new generation networks. In distributed service hosting environments, service mobility allows satisfaction of requirements like: efficient management of available resources, computational load balancing, service continuity even in presence of critical conditions. Service Switching aims at extending the concept of virtualization to network services by decoupling service execution environments and their physical location [5]. Service instances in a Service Switching environment may be dynamically migrated across geographically dispersed datacenters, to achieve more efficient utilization of both network and computing resources. The Service Switching paradigm allows creation and management of *Service Execution Environments* across different datacenters with minimal impact on service continuity.

The architectural implementation of the Service Switching paradigm is centered around a main component, that we call *Service Switch*. Such a component is a network node that, in addition to the plain packet and/or flow switching capabilities, has more advanced features, including the ability to forward packets towards migrated Service Execution Environments. Service Switches can be located both at the edges of a network and in its core. Deployment of Service Switches in the core of the network of course requires cooperation of Internet Service Providers, but allows faster reconfiguration and migration of services.

Our current implementation of the Service Switching model relies on a combination of system-level virtualization technologies and of the Mobile IP model. In the following we firstly introduce a brief description of Mobile IP, and then the Service Switching architecture customized for the NEPTUNE-IaaS context.

IP version 4 assumes that the IP address of a node uniquely identifies its point of attachment to the Internet: a node must be located on the network indicated by its IP address in order to receive datagrams which are destined to it. IP Mobility Support (or Mobile IP) provides a mechanism which allows Mobile Nodes to change their point of attachment to the Internet without changing their IP address [6]. This mechanism relies on two intermediary entities: the *Home Agent* and the *Foreign Agent*. The role of the Home Agent is to maintain current location information of the mobile node, and to re-transmit all the packets addressed to the Mobile Node through a tunnel to the Foreign Agent to which the Mobile Node is currently registered. The role of the Foreign Agent, in turn, is to deliver datagrams to the Mobile Node.

Service Switching allows services to be deployed at different geographic locations, each of which hosts a cluster of physical machines. A physical cluster is connected to the Internet through a special router, that we call *Edge Service Switch*. In the context of NEPTUNE-IaaS we are interested in transparently migrate a collection of related Virtual Machines (a Virtual Infrastructure, according to the definition we gave in Section 2). When a Virtual Infrastructure is deployed for the first time, it is associated to one of the available datacenters. This allocation choice assigns one or more public IP addresses to the Entry Points of the Virtual Infrastructures. These IP addresses will be kept for the entire lifecycle of the Virtual Infrastructure, even in case of migration. Such IP addresses are referred to as the Virtual Infrastructure's *Home Addresses*. The Edge Service Switch located at the edge of the datacenter in which the Virtual Infrastructure is initially deployed, will be referred to as the Virtual Infrastructure's *Home Service Switch*. An Edge Service Switch not only behaves as a normal IP edge router, forwarding incoming packets to the VMs hosted in the cluster and outgoing packets to a next hop router according to its current routing table, but it also implements specific traffic flow readdressing mechanisms to support service migration. Such mechanisms have been derived as extensions of the classical Mobile IP model. A generic end user terminal accessing a service will be referred to as *Correspondent Node*.

Making the simplistic assumption that a Virtual Infrastructure presents a unique Entry Point, in order to access a given service, a Correspondent Node sends packets to this latter, using the VI's Home Address as IP Destination Address. Incoming packets will be processed by the VM's Home Service Switch. In case a Virtual Infrastructure had to be migrated to a different datacenter, the Virtual Infrastructure's Home Service Switch creates an entry in its *Mobility Binding Table* (MBT in short) that contains information about the Entry Point of the migrated Virtual Infrastructure. The MBT keeps the association between the VI's Home Address and the corresponding *Care-of Address*. Such Care-of Address is the IP address of the Edge Service Switch associated to the datacenter hosting the migrated Virtual Infrastructure, that we may call the Virtual Infrastructure's *Foreign Service Switch*. Migration of a Virtual Infrastructure is performed through a procedure that consists in updating the Home Network's MBT and in managing the migration of all the VMs belonging to the Virtual Infrastructure. Concerning the dataceneter that hosts the migrated Virtual Infrastructure, apart from the configuration of the Foreign Service Switch, no other settings are needed. Migrated VMs keep using their own VI's Home Address as IP source address for outgoing packets, and Correspondent Nodes, being unaware of the migration, keep sending packets to the Virtual Infrastructure's Home Address. Once these packets reach the Home Service Switch, this latter forwards them to the Foreign Service Switch, by encapsulating such packets in a point-to-point tunnel (figure 4). The Foreign Service Switch, in turn, de-tunnels the incoming packets and delivers them to the migrated VM. As it happens in the Mobile IP scheme, reverse traffic is sent by the migrated VM directly to the Correspondent Nodes.

4 Optimal VM Allocation in a Datacenter

One of the key steps in the Virtual Infrastructure deployment process is the mapping of Virtual Machines onto the physical resources of the target datacenter.

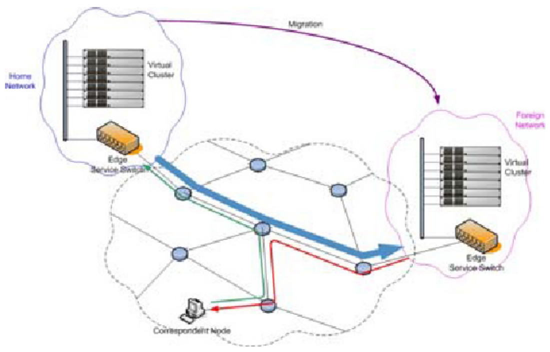


Fig. 4. Tunneling mechanism implemented on the edge

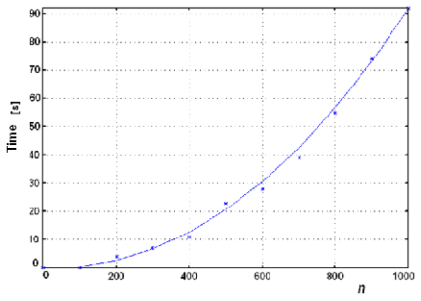


Fig. 5. LK best mapping solution times

This problem is known in literature as the *network testbed mapping problem* [7]. Due to its complexity, the challenge is to find a *good* solution in *acceptable* computational times. Our approach to manage complexity consists in splitting the mapping problem in two sub-problems: *topology partitioning* and a *partition mapping*.

Several graph partitioning algorithms have been proposed in the literature. An algorithm that provides good results with reasonable times of calculation is the Lin-Kernighan (LK) heuristic algorithm [8]. Theoretical complexity of LK is $O(n^2 \log n)$. We implemented this algorithm in JAVA to evaluate its applicability to cluster environments and to assess its performance. A first test was performed to estimate the solver execution time while varying number of nodes in the graph. Size of the matrix was varied between 100x100 and 1000x1000 with steps of 100. The graph was been partitioned into subsets of cardinality equal to 5 while non-zero elements incidence for considered matrix were 2%. Computational times represented in Figure 5 were calculated by using a system equipped with 2 GB of RAM and an Intel CPU T2250 running at 1.73 GHz.

Our algorithm implementation requires that once found a minimum cost solution, the procedure is restarted with a new initial solution. After running 5 iterations the algorithm stops and returns the minimum cost solution. This test

Table 1. Tests organization

	arc/nodes=4	arc/nodes=6	arc/nodes=8
Matrix 20x20	100run	100run	100run
Matrix 100x100	100run	100run	100run
Matrix 400x400	100run	100run	100run

Table 2. Tests results

	i*=1	i*=2	i*=3	i*=4	i*=5
Matrix 20x20 arc/nodes=4	47	21	18	7	7
Matrix 20x20 arc/nodes=6	52	21	13	8	6
Matrix 20x20 arc/nodes=8	43	27	14	6	10
Matrix 100x100 arc/nodes=4	23	21	21	23	12
Matrix 100x100 arc/nodes=6	23	21	28	20	8
Matrix 100x100 arc/nodes=8	18	16	26	18	22
Matrix 400x400 arc/nodes=4	18	24	16	18	24
Matrix 400x400 arc/nodes=6	20	24	18	16	22
Matrix 400x400 arc/nodes=8	22	28	14	12	24

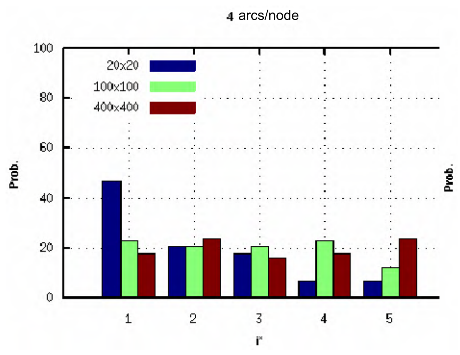


Fig. 6. Four-arcs/node case

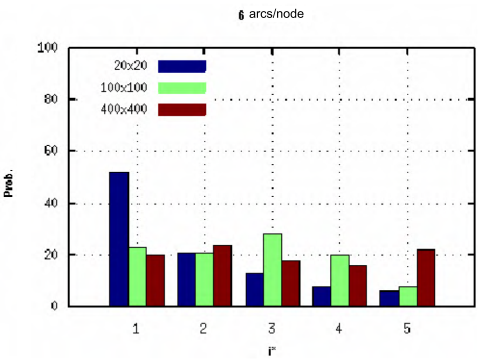


Fig. 7. Six-arcs/node case

highlights the relationship between the iteration $i*$ at which the optimal solution is found with the size and density (arcs/nodes ratio) of the matrix.

Virtual links and physical links bandwidths have been respectively fixed at 10 and 100. Matrices are generated randomly and before subjecting a matrix to the solver, it is verified that each node has at least one connection and that the sum of the costs associated to all the outgoing arcs from one same node does not exceed 90% of the physical connections bandwidth. Tests organization is shown in Table 1, while tests results are shown in Table 2.

Results for the case 4 arcs/node and 6 arcs/node are further shown in Figure 6 and in Figure 7.

This test demonstrates that for matrices of small size (20x20), our solver returns in almost 50% of the cases the least-cost solution at the first iteration.

When the matrix increases in size (100x100 and 400x400), the probability of finding good solutions at the first iteration is lower. In these cases, better results could be obtained by running more iterations, but the rapid increase of computational times does not encourage this approach. The Lin-Kernighan algorithm does not guarantee that it is always possible to find an admissible solution, so it could happen that the found solution does not meet the admissibility constraints. However, in our tests, the solver always returned an acceptable solution.

5 Related Work and Conlcusions

In the last few months the term "Cloud computing" is transforming from a buzzword into real world engineering solutions and commercial products. In this paper we mention two established solutions that have some features in common with NEPTUNE-IaaS: Amazon EC2 and Eucalyptus.

Amazon's *Elastic Compute Cloud* (EC2) [9] is an IaaS commercial system that first introduced the utility computing model, where computation, storage and bandwidth resources are rent on an as-needed basis. As well as NEPTUNE-IaaS, EC2 is based on Xen. Users select an Amazon Machine Image (AMI), including the machine's software configuration from a set of AMIs proposed by Amazon, or create a new one from scratch. To each AMI instance (i.e. a Xen Virtual Machine) is associated an "instance type" that defines the resources of the machine in terms of CPU, RAM, HD. Resources are paid on a consumption basis: a machine is paid for each hour of activity, bandwidth is paid per-gigabyte of traffic and so on. Amazon provides two ways to access EC2 services: via a web interface or through web services. A complete set of tools and programming libraries are provided to access these service.

Eucalyptus [10] is an open-source cloud-computing framework, built to be interface-compatible with Amazon EC2: users can interact with Eucalyptus using same tools and interfaces that they use with Amazon EC2. Because the main goal of Eucalyptus is to provide a common open-source framework that enables researchers to do experiments and studies, even by replacing or modifying the implementation of system modules, the system is based on three components, each with a well defined Web-service interface. The software architecture is hierarchical: the base level is composed by Instance Managers (IM), responsible to manage virtual machines running on top of a physical machines, the middle layer contains Group Managers (GM), each of which manages a set of IMs residing on the same physical subnet. The top layer is the Cloud Manager (CM), that manages all the GM making high-level scheduling decisions and represents the entry-point to Eucalyptus for users as well as for administrators.

NEPTUNE-IaaS has some features in common with both EC2 and Eucalyptus, but also some important differences. In particular, we want to highlight that NEPTUNE-IaaS provides tools to interactively design virtual networked infrastructures and supports transparent and efficient migration of infrastructures across geographically dispersed datacenters. NEPTUNE-IaaS is an ongoing project, whose future development include more complex management procedure to handle migration of complex virtual infrastructures in a reliable way. Integration of NEPTUNE-IaaS with storage services, such as those provided by Amazon's S3 are also being investigated.

Acknowledgements

The research leading to these results has been partially funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°224263-OneLab2.

References

1. Wikipedia: Cloud computing, http://en.wikipedia.org/wiki/Cloud_computing
2. Di Gennaro, P., Bifulco, R., Canonico, R., Ventre, G.: Neptune: Network emulation for protocol tuning and evaluation. Poster presented at SIMUTOOLS'09, Rome (March 2009)
3. 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: Procs. of the 19th ACM Symposium on Operating Systems Principles, SOSP'03., pp. 164–177 (2003)
4. RedHat, et al.: LibVirt virtualization API, http://www.libvirt.org
5. Manetti, V., Canonico, R., Ventre, G., Stavrakakis, I.: System-level virtualization and mobile ip to support service mobility. In: Proceedings of the International Workshop on Design, Optimization and Management of Heterogeneous Networked Systems (DOM-HetNetS'09) (September 2009)
6. Perkins, C., et al.: IP mobility support (1996)
7. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. SIGCOMM Comput. Commun. Rev. 33(2), 65–81 (2003)
8. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. The Bell system technical journal 49(1), 291–307 (1970)
9. Amazon: EC2, http://aws.amazon.com/ec2/
10. Eucalyptus Systems: Eucalyptus web site, http://www.eucalyptus.com/

Performance Measurement of a Private Cloud in the OpenCirrus$^{\text{TM}}$ Testbed

Christian Baun and Marcel Kunze

Steinbuch Centre for Computing,
Karlsruhe Institute of Technology,
Herrmann-von-Helmholtz-Platz 1,
76344 Eggenstein-Leopoldshafen, Germany
`baun,kunze@kit.edu`

Abstract. Cloud computing realizes the advantages and overcomes the restrictions of the grid computing paradigm. Elastic infrastructures can be easily created and managed by cloud users. In order to accelerate the research on data center management and cloud services the OpenCirrus[1] Research Testbed has been started by HP, Intel and Yahoo!. Although commercial cloud offerings are proprietary, an Open Source solution exists in the field of IaaS with Eucalyptus. This paper examines the I/O and CPU performance as well as the network transfer rate of cloud computing infrastructures implemented with Eucalyptus in contrast to Amazon EC2/S3.

1 Cloud Computing – An Upcoming Trend in IT

During the last years with the support of public funding, grid computing evolved from a computer scientists' field of research to a common working environment for scientific disciplines like physics, medicine and meteorology. A grid definition from Ian Foster and Carl Kesselman, summarizing the focus of grids is:

> A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. [1]

At the same time another trend became imminent in the commercial IT sector: Cloud computing [2] aims at consolidating hardware and software resources in large data centers. Cloud computing realizes the advantages and overcomes the restrictions of the grid computing paradigm [3].

All resources are marketed by providers as a service over the Internet based on a utility model. In the world of cloud computing dynamically scalable (elastic) infrastructures can easily be created and managed by the user. Only the consumed resources are accounted following the pay-as-you-go principle. Many different cloud definitions exist and according to our understanding we want to phrase the following short and concise definition:

[1] Open Cirrus is a trademark of Yahoo! Inc.

> Building on compute and storage virtualization, and leveraging the modern Web, cloud computing provides scalable, network-centric, abstracted IT infrastructure, platforms, and applications as on-demand services that are billed by consumption. [4]

Based on this definition we compare grids and clouds. Both technologies focus on IT resources and try to provide an user friendly, inexpensive and pervasive access to these resources over the internet.

The actual situation is that most grid infrastructures consist of geographically distributed, heterogeneous resources without central control, are best suited for special application domains like high energy physics, are publicly funded and use well developed and maintained middleware systems.

In contrast to grids, most cloud infrastructures consist of one or few data centers under central control, are well suited for generic applications, have commercial business models and use proprietary middleware systems. A strong advantage of clouds is the more comfortable usability, as the ownership of resources is granted to the service consumer.

1.1 Everything as a Service

When talking about cloud computing it must be kept in mind that different technical types of cloud services exist.

- **Infrastructure as a Service** (IaaS) implements an abstract view towards the hardware (servers, network,...) and allows to run virtual instances of servers without the need to directly access the bare metal.
- **Platform as a Service** (PaaS) takes the level of abstraction further. PaaS appears as a virtual appliance and makes it simple to scale from a single server to many. Here, the user has no need to worry about the operating system, fundamental software and related application software packages.
- **Software as a Service** (SaaS) provides enterprise quality software (complete applications) to be consumed as a utility.

Cloud computing has the potential to radically change the way IT services are implemented and managed. Project and business funds can be spent to support the core business rather than spending it for IT infrastructure. As they have resource ownership, cloud users are free to run the operating systems, infrastructures, applications and programming languages of their choice. The flexibility of cloud computing has its origin in the combination of virtualization technologies with web services.

2 The OpenCirrus Research Testbed

In July 2008 the OpenCirrus[2] project was announced by HP, Intel and Yahoo!. OpenCirrus aims to build an open, internet-scale global testbed for cloud computing research focusing on data center management, cloud services, systems

[2] `http://opencirrus.org`

and application level research. OpenCirrus is a loose federation of three sponsors which are HP Labs, Intel Research and Yahoo! At the moment there are three also academic partners: The University of Illinois at Urbana-Champaign (UIUC), the Singapore Infocomm Development Authority (IDA) and the Karlsruhe Institute of Technology (KIT).

OpenCirrus consists of six sites initially, hosted by the sponsors and partners, each equipped with 1000–4000 CPU cores and 1 Petabyte of data store.

The basis of the OpenCirrus research testbed is formed by the Physical Resource Sets (PRS) [5]. These provide logical mini-datacenters to the researchers and isolate the experiments from each other. Inside a PRS ensembles of physical nodes are allocated and isolated via virtual local area networks (VLAN) using already existing software like Emulab, a network emulation testbed from the University of Utah and HP Opsware, a tool for server provisioning, configuration and management.

The PRS are the basis to implement Virtual Resource Sets (VRS). The VRS abstract from physical resources by the introduction of a virtualization layer. The virtualization concept applies to all IT aspects like CPU, storage, networks and applications. The main advantage of VRS is the potential to create IT services exactly fitting customers varying needs. IT services can be deployed on demand by automated resource management. Service levels can be easily guaranteed and live migration of services is possible. As a consequence capital expenditures and operational expenditures are both reduced.

The VRS are implementing Infrastructure as a Service (IaaS) because they provide compute, storage and networking services.

OpenCirrus strongly differs from other cloud computing testbeds because it supports both, system and application level research. In contrast to cloud infrastructures like Amazon Elastic Compute Cloud (EC2)[3], Amazon Simple Storage Service (S3)[4] and Google AppEngine[5], all software layers and the hardware itself can be accessed and adapted by the OpenCirrus researchers. Intel platform features like Intel Data Center Management Interface (DCMI) and Node Manager (NM) that support cloud computing could also be utilized.

3 Eucalyptus

A cloud service that will be examined by KIT in the OpenCirrus research testbed is the cloud computing infrastructure service Eucalyptus. Eucalyptus[6] is an Open Source software developed at the University of California, Santa Barbara, implementing cloud computing on university compute clusters. EUCALYPTUS stands for Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems. It implements Infrastructure as a Service (IaaS) while giving the user the ability to run and control virtual machine instances (Xen) deployed

[3] `http://aws.amazon.com/ec2/`
[4] `http://aws.amazon.com/s3/`
[5] `http://code.google.com/appengine/`
[6] `http://open.eucalyptus.com`

across a variety of physical resources [6]. The interface is compatible with EC2 which is the most popular IaaS, and Eucalyptus includes Walrus, a basic implementation of the S3 interface and a block storage service that is interface compatible with Amazon EBS[7].

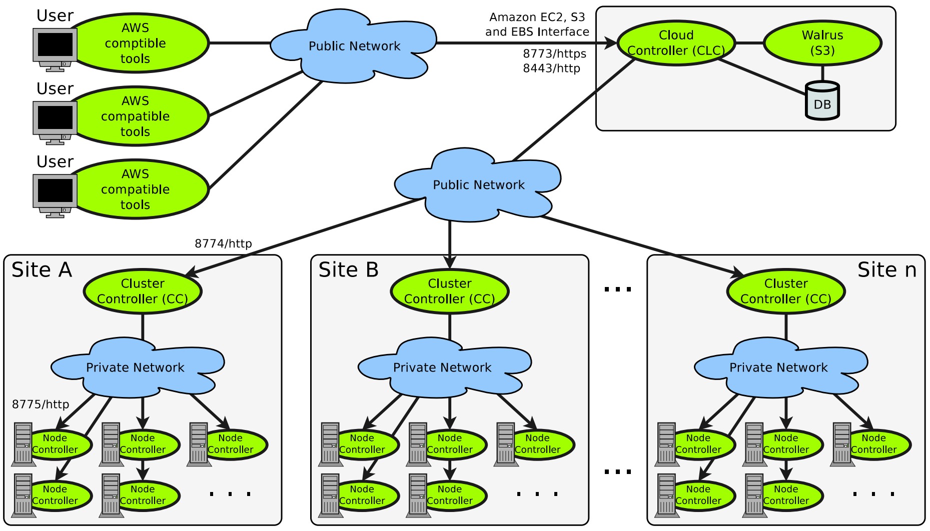


Fig. 1. Structure of Eucalyptus

With Eucalyptus it is possible to build up a private cloud that can be controlled by the same tools known to work with Amazon EC2 and S3. Examples are the ElasticFox EC2 plugin for the Firefox browser or s3cmd utilities for S3 storage management. Eucalyptus has the potential to help establish an open cloud computing infrastructure standard. The main components are the Cloud Controller (CLC), Cluster Controller (CC) and Node Controller (NC) [7]. The NC runs on every node in the cloud as well as a Xen-Hypervisor[8] or KVM[9]. The NC provides information about free resources to the CC. The CC schedules the distribution of virtual machines to the NC and collects (free) resource information. The CLC collects resource information from the CC and operates like a meta-scheduler in the cloud. Figure 1 shows the structure of Eucalyptus including CLC, CC and NC.

For preliminary testing an Eucalyptus R&D cloud installation has been set up at KIT running Eucalyptus 1.4. This installation is used for gaining experience with Eucalyptus and several performance tests. The R&D cloud consists of:

– 2x IBM Blade LS20 (2x Single Core Opteron at 2.4 GHz, 4 GB RAM)
– 2x IBM Blade HS21 (2x Dual Core Xeon at 2.33 GHz, 16 GB RAM)

[7] `http://aws.amazon.com/ebs/`
[8] `http://www.xen.org`
[9] `http://www.linux-kvm.org`

The LS20 Blade is from 2005 and the HS21 Blade from 2006. One LS20 acts as CLC, CC and NC. This consolidation leads not to performance issues because of the small number of NCs in this installation. The other LC20 and the two HS21 are NCs.

While the performance of an Eucalyptus private cloud depends on the hardware available, it is interesting to see the performance key data of this Eucalyptus installation, compared to Amazon EC2/S3. The focus of the performance benchmarks was storage, network and CPU performance.

3.1 Storage Performance of Eucalyptus

In order to compare the storage performance of Amazon S3 and Eucalyptus 1.4 with Walrus, the benchmark tool `Bonnie++`[10] was used. This software measures the rate of sequential output and input. The measurements in Figure 2 are sequential output (per character, per block and rewrite) and sequential input (per character and per block).

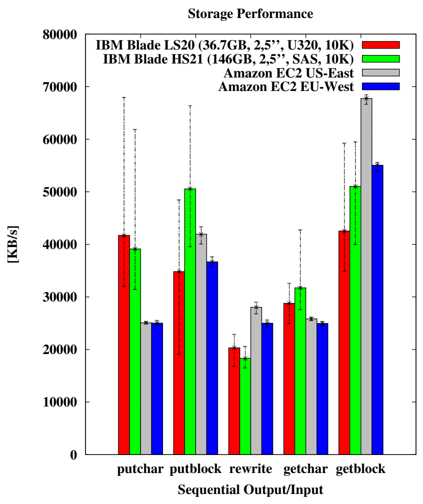


Fig. 2. Storage Performance of Amazon S3 and Eucalyptus

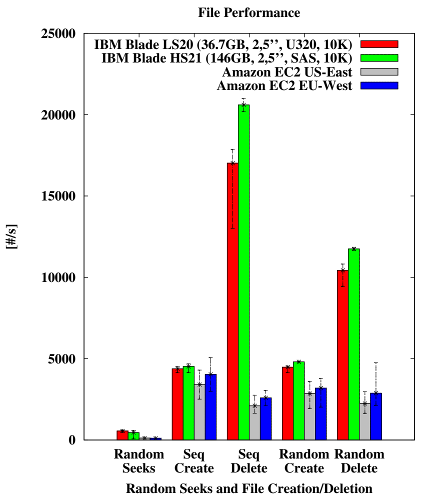


Fig. 3. Performance of Random Seeks and File Creation/Deletion for Amazon S3 and Eucalyptus

The RAM of the Eucalyptus NCs was reduced to overcome memory caching effects. The storage performance of Eucalyptus depends on the features of the storage subsystem. For these tests, the write performance of Eucalyptus, using a modern SAS hard disk with 10000 RPM (revolutions per minute) is faster than Amazon S3. The performance for read in contrast is faster at the Amazon sites.

[10] `http://sourceforge.net/projects/bonnie/`

Bonnie++ also measures the performance for random seeks and especially file creation (Figure 3). Both is faster with Eucalyptus. A possible explanation for these measurements is that Eucalyptus stores the data at the NCs locally. It is unknown whether data stored by Amazon S3 is located near or far the EC2 instances. The reason why file deletion at this Eucalyptus installation performs that much better compared to Amazon S3, remains unclear.

While testing the storage performance of Eucalyptus, the instances were running alone on the blade servers to avoid interferences. We cannot make any assertion about the workload of the Amazon S3 system and the load of the physical server the EC2 instances were hosting at Amazon during our testing.

3.2 Network Transfer Rate

The network transfer rate inside and between the Eucalyptus site and the Amazon EC2 sites was measured at a working day (July 2th 2009) with `iperf`.

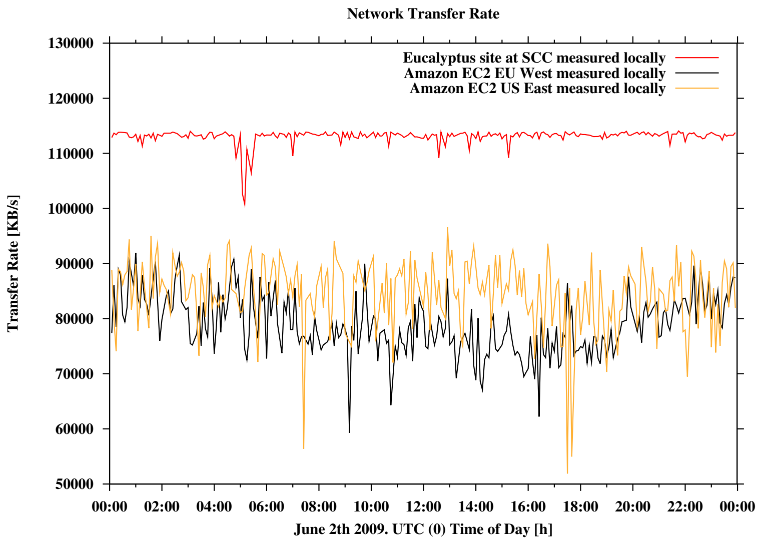


Fig. 4. Network Transfer Rate (1)

The strong in-house network transfer rate (Figure 4) is not surprising because of the 1000 Mbit/s Ethernet. But it is evident that the network transfer rate to the Eucalyptus infrastructure is much more constant in contrast to Amazon EC2. The network transfer rates inside the Amazon EC2 US East and EU West sites imply that there is 1000 Mbit/s Ethernet also used, but with a higher workload.

Figure 5 shows, that the network transfer rate from Karlsruhe to Amazon EC2 EU West is approximately twice as much better compared to Amazon EC2 US East. The network transfer rate to Eucalyptus over the German national research and education network (DFN) is more constant compared to Amazon

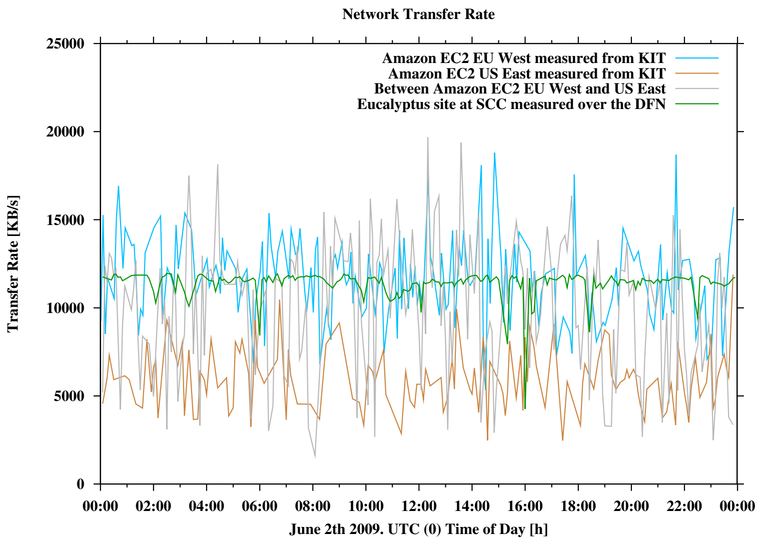


Fig. 5. Network Transfer Rate (2)

EC2. This is also not surprising. An interesting outcome is that peaks of the network transfer rate to Amazon EC2 EU West are much better compared to the connection to the Eucalyptus site in Karlsruhe over the DFN.

3.3 Network Latency

To examine the potential for using Ecalyptus and public cloud infrastructures at all for HPC, the network latency was measured. The results in Table 1 show that HPC in the cloud over institutional/geographical borders is impossible.

For MPI-Jobs where every task computes a few seconds like Monte Carlo methods it is possible to use cloud infrastructures. The network latency inside Amazon EC2 is poor and a surprising outcome is that the network latency between Amazon EC2 EU West and Amazon EC2 US East is better than inside the Amazon sites.

Table 1. Network Latency

`ping ip -f -c 10000`	time [ms]	min. Round-Trip-Time [ms]	avg. Round-Trip-Time [ms]	max. Round-Trip-Time [ms]
EC2 EU West from KIT	138262	27.943	28.192	59.399
EC2 US East from KIT	137014	92.839	93.154	118.853
inside EU West	146447	87.493	90.069	145.109
inside EC2 US East	147380	87.527	92.266	115.461
EC2 EU from EC2 US	138451	88.260	90.776	144.078
Eucalyptus at SCC via DFN	131145	15.093	15.197	29.863
inside Eucalyptus at SCC	2064	0.125	0.146	0.806

3.4 CPU Performance

To compare the CPU performance of Amazon EC2 and Eucalyptus 1.4, the Linux Kernel was compiled for benchmarking. All available Amazon EC2 instance types (see Table 2) were tested.

Table 2. Amazon EC2 Instance Types

`m1.small`	(Small Instance)	1.7 GB RAM	1 virtual Core
`c1.medium`	(High-CPU Medium Instance)	1.7 GB RAM	2 virtual Cores
`m1.large`	(Large Instance)	7.5 GB RAM	2 virtual Cores
`m1.xlarge`	(Extra Large Instance)	15 GB RAM	4 virtual Cores
`c1.xlarge`	(High-CPU Extra Large Instance)	7 GB RAM	8 virtual Cores

For `m1.small` instances, the single virtual core is equivalent to one EC2 Compute Unit. One Amazon EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. This is also the equivalent to an early 2006 1.7 GHz Xeon processor.[11] The virtual cores of the `m1.large` and `m1.xlarge` instances are equivalent to two EC2 Compute Units each. The virtual cores of the `c1.medium` and `c1.xlarge` instances are equivalent to 2.5 EC2 Compute Units each.

Eucalyptus even provides five instance types (see Table 3) following the identical naming scheme than EC2.

Table 3. Eucalyptus Instance Types

`m1.small`	(Small Instance)	128 MB RAM	1 virtual CPU
`c1.medium`	(High-CPU Medium Instance)	256 MB RAM	1 virtual CPU
`m1.large`	(Large Instance)	512 MB RAM	2 virtual CPUs
`m1.xlarge`	(Extra Large Instance)	1 GB RAM	2 virtual CPUs
`c1.xlarge`	(High-CPU Extra Large Instance)	2 GB RAM	4 virtual CPUs

For CPU benchmarking, the time needed to compile Linux Kernel 2.6.29.3 with 1, 2, 4 and 8 threads was measured. The results in Figure 6 show that additional RAM and CPUs are leading to a significant performance boost when using more threads. For none of the instance types more than 8 threads lead to better results.

The measurements in Figure 6 also show our Eucalyptus infrastructure performs approximately twice better for `m1.small` instances compared to Amazon EC2. This is not surprising because the CPU differs and due to the fact that while testing the CPU performance of Eucalyptus the instances were running alone on the blade servers to avoid interferences. We cannot make any assertion about the load of the physical server the EC2 instances were hosting at Amazon

[11] `http://aws.amazon.com/ec2/instance-types/`

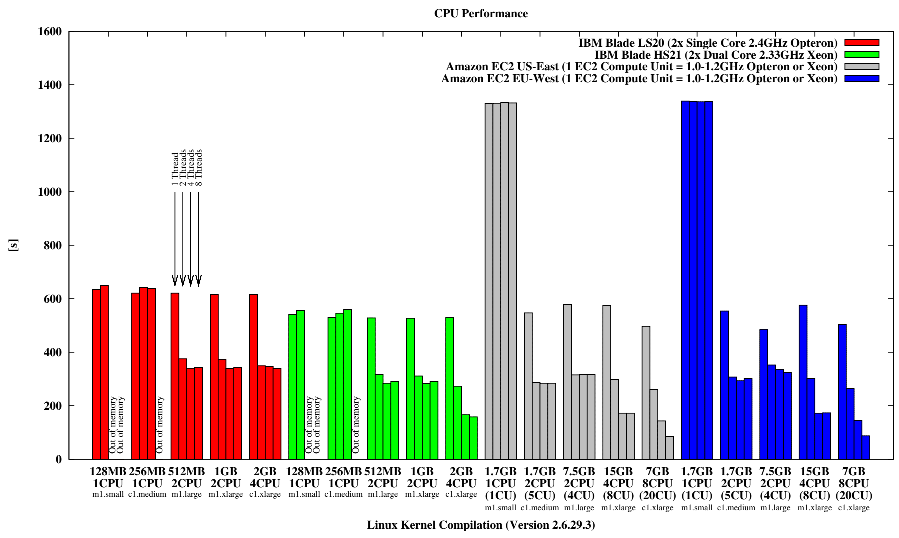


Fig. 6. CPU Performance measurement via Linux Kernel compilation

during our CPU testing. But it is likely that the Amazon servers storing `m1.small` instances have to share their resources between lots of instances thus reducing the CPU performance radically.

Using more threads than virtual/physical CPUs/cores available is not leading to a performance boost because of the thread context switching overhead. The reason why using more than 2 threads at `c1.xlarge` with Eucalyptus at the IBM LS20 is not leading to a significant performance boost is because the LS20 has only two single core CPUs. The IBM HS21 has two dual core CPUs and therefore using 4 threads at `c1.xlarge` leads to an performance enhancement.

The CPU performance measurements strongly depend on the workload of the physical machines. For Eucalyptus, the instances were running alone on the blade servers to avoid interferences but it is impossible to make any assertion about the workload of the physical servers inside the Amazon sites.

4 Further Steps

Currently a second Eucalyptus R&D cloud is set up at KIT with 5x HP Blade ProLiant BL2x220c. Each blade includes two server nodes (2x Intel Quad-Core Xeon at 2.33 GHz, 16 GB RAM). This system is running Eucalyptus 1.5.2 with AppScale 1.1. The purpose of this new installation is to gain experience to operate the OpenCirrus KIT site with virtualization services based on 2656 Nehalem cores in 332 HP T2 servers starting in autumn 2009.

5 Conclusion

Cloud computing allows flexible and elastic resource provisioning. The high degree of automation and the large economies of scale make it attractive for both, academia and business, pathing the way from manufacture towards the industrialization of IT.

OpenCirrus offers interesting R&D opportunities for cloud systems research and application development.

With Eucalyptus, an Open Source implementation of the perhaps most popular IaaS offering of Amazon is available, representing a first step towards the creation of a cloud standard. Although Eucalyptus is a new development the software performs sufficiently stable and when using up to date hardware the users have no need to fear a lower performance as compared to Amazon. The performance that can be achieved with Eucalyptus depends on the physical servers and their workload.

With commodity hardware and Open Source software, a private cloud can be build up providing the same functionality and better performance compared to the most popular public clouds.

References

1. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
2. Buyya, R., Yeo, C.S., Venugopal, S.: Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In: HPCC'08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications (2008)
3. Bégin, M.E.: An EGEE Comparative Study: Grids and Clouds – Evolution or Revolution, CERN (2008)
4. Baun, C., Kunze, M., Nimis, J., Tai, S.: Cloud Computing: Web-basierte dynamische IT-Services. Springer, Heidelberg (2009)
5. Campbell, R., Gupta, I., Heath, M., Ko, S., Kozuch, M., Kunze, M., Kwan, T., Lai, K., Yan Lee, H., Lyons, M., Milojicic, D., O'Hallaron, D., Soh, Y.C.: Open Cirrus^{TM} Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research. In: HotCloud'09: Workshop in Hot Topics in Cloud Computing (2009)
6. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The Eucalyptus Open-source Cloud-computing System. In: CCA'08: Proceedings of Cloud Computing and Its Applications workshop (2008)
7. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. In: UCSB Computer Science Technical Report Number 2008-10 (2008)

Providing Grid Services Based on Virtualization and Cloud Technologies

Javier López Cacheiro, Carlos Fernández, Esteban Freire, Sergio Díaz, and Alvaro Simón

CESGA, Santiago de Compostela, Spain
egee-admin@cesga.es
http://www.cesga.es/

Abstract. CESGA is operating a totally virtualized grid infrastructure that supports several production sites for different grid projects (EGEE, EELA, int.eu.grid (I2G), Ibergrid, and other regional grid projects) as well as several development sites created on demand to test new middleware releases both for EGEE certification and pre-production activities. The final architecture that results from several years of development is described showing how to apply modern virtualization solutions like Xen hypervisor to migrate from an entire physical cluster to virtual machines. Thanks to a collaboration with FORMIGA's project the infrastructure also includes resources from computer labs. Worker Node VMs are automatically started following a pre-defined schedule that guarantees that the computers are not in use. Extensive benchmarks, including two of the most used applications at our supercomputing center, have been performed to quantify the performance loss of the virtual infrastructure. Currently cloud computing technologies are being explored as a way to improve the service deployment process in our platform.

1 Introduction

CESGA has been providing support for an increasing number of new grid users and projects for the last years. At present CESGA is supporting these projects EGEE III, I2G, IBERGRID [1], EELA-II [2], Spanish-NGI and FORMIGA [3].

All these projects are based on the gLite middleware [4] developed inside EGEE, and Spanish NGI also supports Globus 4 Toolkit. Apart from supporting these grid projects, CESGA collaborates testing new middleware releases for EGEE certification and pre-production activities. These activities require a continuous deployment of updated and new services.

The main reason to migrate CESGA grid services to virtual machines was the need to support new hardware running old operating systems (OS), like Scientific Linux 3 (SL3) which was required by gLite in 2007. Thanks to the hypervisor new hardware (network interfaces, hard disks, etc) can be used in a transparent way by the guest OS. After new infrastructure deployment the next step was to integrate it with computer labs to take advantage of unused CPU resources. This task was done by FORMIGA project, quite similar to BOINC [5] project, but focused on computer labs *gridification* for EGEE.

VMs performance is also important to determine migration pros and cons. Along the past four years numerous papers have been published analysing VM performance but only a few of them consider real-world parallel applications when evaluating performance. For this reason, in this paper selected benchmarks have been included to show specific CESGA virtual platform performance, that it is relevant for most of our users. From previous papers, perhaps the most well-known it is the one by Walker [6] comparing MPI and OpenMP running over an NCSA cluster and Amazon EC2, the results shows up to 21% degradation for OpenMP and up to 1000% degradation for MPI running computational fluid dynamics application (SP) using the Beam-Warming approximate factorization method. On the other hand one of the most exhaustive studies is the one by Bavelski [7] and presented in his final thesis. I/O drive bound tests revealed 5 to 10 times worse performance over Xen virtual machines performance being even worse in the case of purely hard drive-bound. Ho [8] has also done an extensive evaluation of Xen performance using Intel MPI Benchmark Suite and UnixBench including checkpointing evaluation. Both full-virtualization and para-virtualization are considered in his study, it is found that in the worse case VMs are up to 2 times slower. Finally, Huang [9] has written a dissertation about high performance network I/O in VMs over modern interconnects evaluating MPI performance and RDMA based migration of VM. The dissertation includes several parallel benchmarks, including NAS Parallel Benchmarks [10]. In this paper selected benchmarks have been included to show specific CESGA virtual platform performance, that it is relevant for most of our users.

This paper is structured as follows: in Section 2, CESGA virtual infrastructure is presented. In Section 3, impact on performance of virtualization is evaluated. Section 4 describes the use of the emergent cloud technology to improve resource management. Finally, Section 5 provides a summary of the main conclusions of this work.

2 CESGA Grid Infrastructure

Two years ago the migration of our worker nodes (WN) to virtual machines (VM) was started mainly due to limited hardware support in SL3, the only OS supported by gLite at that time. In these sections the main characteristics of CESGA infrastructure are presented: virtualization, shared batch system and WN and the reuse of computer labs resources.

2.1 Virtualization

All grid services at CESGA are now running under a totally virtualized infrastructure which allows to easily support new projects on demand. When a new physical machine arrives it is quickly configured and installed using a specific kickstart script which automatically installs a new Xen [11] dom0 server from a local repository using *Preboot eXecution Environment* (PXE). Each new deployed dom0 can run several VM with different OS like SL3 or SL4, depending on the service requirements. When it is needed to configure a new service, a *golden-copy* from our local VM repository is used to deploy it into any available dom0 and then, if it is based on gLite, it is configured using Yaim and CESGA global site-info.def in a few minutes.

The main advantages of our virtualized grid infrastructure are:

- Better resource utilization: New services can be installed fast and there is no need to spend money on new hardware. It can be a solution for sites and users who increasingly demand more services.
- Power saving: Several grid services are consolidated in a single server (up to 8 services depending on the requirements of the service) reducing the number of servers required and therefore reducing overall power consumption in the datacenter.
- Easy replication: The deployment of a VM from an existing template or golden copy is done in just a few minutes.
- Load balancing: If a VM requires more physical resources, it can be given more resources or migrated to another dom0.
- Fault-tolerance: Xen combined with Logical Volume Manager (LVM) offers roll-back possibility using snapshots. In case of failure, a VM can be replicated using our daily backups and started in a different dom0.
- Flexibility: Possibility of using old OS versions, like Scientific Linux 3 with modern hardware.

2.2 Shared Batch System and Worker Nodes

In order to share all the WN among the different grid projects supported at CESGA a shared batch system is required. In our case Grid Engine (GE) batch system [12] is used. gLite middleware requires a Computer Element (CE) for each grid infrastructure (see Figure 1). The batch server is shared using one single GE qmaster server and a shadow qmaster for fault tolerance purposes with our current configuration. Jobs are submitted from different sources but all jobs are collected in a single batch server (qmaster in the GE nomenclature) that distributes them between all the available WN. At this point, a difficulty appears due to the fact that different WNs belonging to different grid projects require a different environment and in some cases even a different middleware version. This issue is solved by the GE JobManager developed at CESGA, it is configured on each CE to load the specific gLite environment for each project.

2.3 Integrating Computer Lab Resources

FORMIGA project has extended the gLite middleware to take advantage of idle resources at computer labs. The project final prototype is already working in several computer labs of the University of Santiago de Compostela and CESGA. These computers run Xen or VMware depending on the OS installed by the computer lab administrator, operating transparently to the user. The virtual machines installed at computer labs are used as WN that communicate

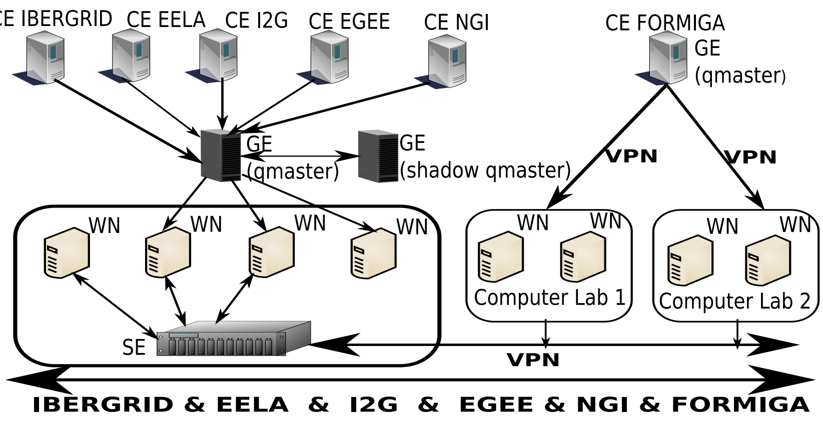


Fig. 1. CESGA grid infrastructure schema with shared WN, GE batch system and SE

with one CE allocated in CESGA securely through a virtual private network (VPN) managed using SSL/TLS for asymmetric encryption between the WN in computer labs and CESGA servers (CE, SE) (see Figure 1). All nodes are interconnected through the VPN, each virtual machine, after starting its network interface with connection to Internet, starts the client openvpn service. For security reasons only computer labs administrators can access directly to running virtual machines, grid non-privileged users are authenticated using their specific X.509 certificate signed by a certificate authority (CA) to execute their jobs. The usage of virtual machines allows an easy deployment of the middleware in the computer labs and permits to migrate jobs between computers, besides the use of Xen on the platform provides flexibility to manage them. The VPN helps to avoid the restrictions of firewall and private networks usually configured in the computer labs.

3 Benchmarks

One of the main concerns about using a virtual infrastructure is the performance loss incurred by the virtualization layer. Not many years ago nobody would believe that a complete production site could be run efficiently under a virtual environment and at that time virtualization technology was useful only for academic or testing purposes due to their performance. With the improvements in virtualization technology and the arrival of Xen, an open source virtual machine monitor (VMM) that implements the concept of paravirtualization, new doors have been opened to implement a production grid infrastructure.

To quantify the performance loss incurred by using virtualization technology, several benchmarks have been performed both using this virtual grid infrastructure and a physical one and shows a general overview of CESGA infrastructure performance. The main results of these benchmarks are presented in this section.

Table 1. VM performance loss by type of benchmark: a comparison with results from previous papers is also included

	CESGA	Previous papers
Disk sequential read	0%-1%	66% [7]
Disk sequential write	4%-26%	63% [7]
Disk random read	1%	83% [7]
Disk random write	34%	67% [7]
CPU performance	0%-22%	0% [8] 2%-3% [7]
OpenMP	1%	7%-21% [6]
MPI	294%	40%-1000% [6]
Network latency	41%	41% [8]
Network bandwidth	15%	30% [9]

In our performance evaluation a system composed of two Dell PE1955 blades with the following characteristics were used:

– 2 x Intel(R) Xeon(R) CPU E5310 @ 1.60GHz QuadCore
– 4GB DDR2-667 RAM, 73.4GB HDD
– Gigabit ethernet: Broadcom 5708
– Dom0: Fedora Core 6 x86_64
– VM: Scientific Linux 4 i386
– Xen: Linux Kernel 2.6.18-1.2798.fc6xen

These two blades are very representative of CESGA configuration where most of the servers are of this type so it will give us a good estimation of the performance of our overall grid infrastructure. Each benchmark was executed three times using these dedicated nodes exclusively, the results shown are the arithmetic average of the measures.

The benchmarks performed can be divided in two groups, synthetic benchmarks where the performance of specific components of the system was evaluated, and application benchmarks where a selection of the most commonly used applications in our supercomputer center has been selected to evaluate their performance in both platforms.

3.1 Synthetic Benchmarks

Several synthetic benchmarks were selected to measure CPU, filesystem and network performance:

– Intel Linpack [13]: CPU performance
– Bonnie++ 1.03a [14]: filesystem performance benchmark tool
– Iozone 3.323 [15]: newer filesystem benchmark tool
– Iperf 2.0.4 [16]: modern alternative for measuring maximum TCP and UDP bandwidth performance

– Effective Bandwith Benchmark (b_{eff}) 3.5 [17]: measures the accumulated bandwidth and latency of MPI jobs, an older version of this tool is part of the well-know HPCC benchmark suite

All those benchmarks were executed using the same parameters both in the virtual system and in real machines. The detailed results can be downloaded from [18].

The file system performance of virtual and physical machines is very similar (less than 1% performance difference) except in the case of the write performance tests where there is an important performance penalty incurred by virtual machines. In the sequential write test the VM suffers a 15% performance degradation with respect to the physical one and in the random write test this degradation reaches 30%, these results are better than previous published works [7] (see comparison Table 1), because newer Xen versions have improved I/O performance. This means that write throughput is seriously impacted in virtual machines and their use for highly intensive I/O applications is not recommended. In our virtual infrastructure this is partially avoided by the fact that an externally exported StorageWorks Scalable File Share (SFS) filesystem [19] is used as the main storage source for the SE machines.

Another important aspect is the CPU performance, because HPC grid and cloud computing users demand fast CPUs for their applications and VMs must provide an adequate performance. It should be mentioned here that CESGA virtual grid services are running in paravirtual machines. This has the disadvantage that the original guest must be modified to use the Xen kernel but offers many advantages in terms of performance. In this case the virtual guest is aware that it is running in a virtualized environment and it communicates directly with the Xen kernel hypervisor reducing the performance penalty incurred by virtualization.

To compare CPU performance, all tests were performed running on x86_64 CPUs (with 51.2 Gflops of theoretical peak) using a x86_64 bit Xen kernel version both for real and VM machines. The synthetic benchmark chosen for these tests was Intel Linpack, both the x86_64 and i386 versions. This is due to the fact that the OS of virtual machines is the i386 version of Scientific Linux 4 and not the x86_64 version because the x86_64 version of gLite still has many bugs for a production environment.

VM performance running x86_64 linpack is exactly the same than real machines, showing basically the same conclusion than previous papers (see Table 1). Unfortunately for i386 binaries the situation is drastically different, VM looses about 22% performance, in dom0 we obtain 37 GFlops but in Xen VM we obtain 29 Gflops running linpack compiled for i386 architecture. This situation may be due to the fact that the Xen hypervisor must translate i386 instructions running into x86_64 kernel. This situation can be critical for user applications compiled for i386 which are executed on VMs with x86_64 kernels, the best option in this case is to recompile user applications for x86_64 architecture or change VM to its x86_64 version.

Finally, network performance was evaluated using iperf and b_{eff}. In this case two dom0s and two VMs were used to measure network bandwidth between them. The result for real machines was a bandwidth of 871 Mbits/sec meanwhile for VMs it decreases to 740 Mbits/sec, this means a 10% loss (about 10%). To measure MPI network performance the algorithm of effective bandwidth (b_{eff}) was executed. The results for real machines were b_{eff}= 53.92 MB/s , Latency= 83.26 μs and for VMs b_{eff}= 42.63 MB/s , Latency= 117.14 μs. This means a 41% loss in MPI latencies and a 15% loss in b_{eff} which greatly impacts performance of communication-bound MPI applications as discussed in the next Section. These results are in line with previous measurements of network latency from Ho[8] but greatly improve the previous measurements of network bandwidth by Huang[9] going from a 30% to a 15% loss as it can be seen in Table 1. This is mainly due to the high performance networking using segmentation off-load introduced in Xen 3.0.3 (Huang[9] used the older version 3.0).

3.2 Application Benchmarks

Synthetic Benchmarks could be a good reference to compare different machines but it is ever better to test *real-life* applications. In this case two of the most used applications in our supercomputing center were selected trying to simulate a real scenario of what it is actually run at CESGA by our users:

- Gaussian G03: Computational chemistry simulation package [20]
- Gromacs 3.3.2: Molecular dynamics simulation package [21]

In Gaussian case was used the serial version program running $Na(H_2O)_4$ S_4 symmetry example (Test339). In this benchmark results were quite similar on both cases, the test was terminated in 910s and 920s running on a real machine and VM respectively. This benchmark clearly demonstrates the analogous throughput between real and VMs when no intensive I/O operations are needed with a performance loss of just 1%. This result shows a great improvement for OpenMP applications compared to those previously obtained by Walker [6] where losses between 7 and 21% were reported depending on the OpenMP application.

The official Gromacs DPPC benchmark was executed using eight MPI processes running on same node. DPPC emulates a phospholipid membrane for a total of 121,856 atoms. VMs are 1% slower running this test and the loss is almost negligible.

One of the most remarkable characteristics of this test was the great difference between real and virtual machines running the same DPPC benchmark over MPI. Two nodes running eight MPI processes in each one were used. On real machines the test finished in 604s, 28% faster than MPI execution using a single node, but on the other side, VMs are much slower, finishing the tests in 2379s (1529s slower that running the same job only using 8 CPUs). This result, 294% performance loss, is better than the results obtained by [6] (see Table 1), where up to a 1000% loss was detected, but it is still too high. The main reason of this significant performance loss is the high dependency on network latencies of the DPPC benchmarks and the fact that, as it has been shown in the previous section, network latency is 41% worse in VMs.

Summarizing, OpenMP applications like Gaussian G03 are well suited to run in our virtual environment with almost no performance loss if the calculations do not perform intensive I/O operations. However communication-intensive MPI applications like our Gromacs example should not be run between VMs since performance can be even worse than that of a single node.

4 Towards Cloud Computing

As discussed above, most of the CESGA Grid infrastructure is based on virtual machines, this adds a lot of flexibility to the architecture, specially when services are necessary in a short time. This infrastructure uses a central repository which stores different virtual machines as Scientific Linux 4/5, Open Suse 9/10, etc. These virtual machines are like dummy boxes without any service, when a new gLite service is needed the dummy virtual machine image is copied by hand from repository to a specific Xen dom0. After that new VM is started on their new location, a new IP is assigned to it and middleware services are installed following the normal process. This procedure saves a lot of work and is very flexible, virtualization allows moving VM images to another location in a few seconds, change their available memory in hot, make image snapshots to recover them from a disaster, change CPU allocated,etc.

Until now virtual machines are started by hand in our virtual grid infrastructure, but using a web service like Eucalyptus, VMs (or instances in *cloud* terms) could be started with a minimal effort running new gLite services, only by clicking in a web interface.

At the time of writing this article, we are testing Eucalyptus in our grid farm. The first tests have been promising, Cloud services do not consume too much memory and Cloud Nodes (CN) and Controllers use secure internal communications with WS-security without efficiency lost. There are still questions to be addressed as fault tolerance if a Node Controller fails, Cluster Controller must check this issue and replace failing instances in a new dom0. Other question is about coexistence between Eucalyptus VMs and other VMs on same CN, at the moment Eucalyptus essentially ignores VMs started outside of its control, but probably these features will be available in next releases.

5 Conclusions

The main advantages of CESGA virtual grid infrastructure are flexibility, improved resource utilization, easy replication, load balancing and fault-tolerance capabilities. In order to share available resources between all the different grid projects supported at CESGA, the infrastructure uses a single batch system shared among all CE that allows to use a common pool of WN for all projects. The modifications required in the WN has been described and could be implemented for other grid projects. Additionally idle resources at computer labs are included in the infrastructure by using the software developed in FORMIGA's project that allows an easy integration of these spare resources with any gLite-based environment.

The results of our benchmarks show that the performance of the virtual infrastructure is very similar to the one of its physical counterpart (less than 1% performance difference) with two exceptions that it is worth to mention: I/O write performance and MPI latency. In the case of I/O write performance a degradation up to 30% could be experienced and, in the MPI latency case, benchmarks show a 40% loss, but the worst throughput result was running Gromacs MPI over virtual machines.

This means that CESGA virtual grid infrastructure is not recommended for running highly intensive I/O applications that rely heavily on random write performance or highly parallel MPI jobs where latency plays an important role. The second limitation does not represent a big problem in a typical grid environment where most of the jobs are not using MPI, but scientists should be aware of this performance penalty if they pretend to migrate MPI applications to cloud computing. To solve this issue, one of the principal virtualization objectives is to improve drivers performance and develop new OS-bypass and VMM-bypass mechanisms to avoid VMs I/O overhead.

In our virtual infrastructure the effects of the limited write performance are partially avoided by the fact that an externally exported SFS filesystem is used as the main storage source for the SE machines.

The results of the selected application benchmarks show that our virtualized infrastructure is valuable for most of our grid users. On the other hand, it should be also stressed that current MPI performance is very poor and it is not recommended to run communication-intensive MPI applications in a multi-node virtualized environment.

Cloud services add a new layer of abstraction, where available resources are managed more easily and grid administrators do not have to worry about searching the pool of resources, cloud does it. To make this possible, new open source projects like Eucalyptus offer us an excellent bridge to convert our Xen based infrastructure into a new cloud.

Acknowledgments

This work was supported by European Comission under the project EGEE-III (INFSO-RI-222667), and by Xunta de Galicia under the projects FORMIGA (07TIC01CT) and G-Fluxo (07SIN001CT).

References

1. Campos, I., Lopez, A., Orviz, P., Gomes, J., Borges, G., Diaz, S., Fernández, C., Lopez, J.: IBERGRID, Grid infrastructure for the Iberian Research Area. In: IBERGRID'09 (2009)
2. Carvalho, D., Duarte, A., Diacovo, R.: The EELA-2 Project e-Infrastructure. In: First EELA-2 Conference (2009)

3. López Cacheiro, J., Cordero Placer, D., Fernández Iglesias, C., Gutiérrez Sanmartín, E., Valin, R., Fernández Sánchez, C., López Cabido, J.I., Rodríguez López, A., Garcia-Loureiro, A., Aldegunde, M., Seoane, N., Pena, T., Cabaleiro, J., Rivera, F.: Formiga/g-fluxo: Adding computer labs to the grid. In: Ibergrid'09 (May 20, 2009)
4. EGEE: gLite, http://glite.web.cern.ch/glite (Last visit: 26-06-2009)
5. Anderson, D.: Boinc: A system for public-resource computing and storage. In: Proceedings 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, EEUU (2004)
6. Walker, E.: Benchmarking Amazon EC2 for high-performance scientific computing. Login 33, 5 (2008)
7. Bavelski, A.: On the Performance of the Solaris Operating System under the Xen Security-enabled Hypervisor. PhD thesis, Linkopings universitet, Department of Computer and Information Science (2007)
8. Ho, C.: Evaluation of Xen: Performance and Use in Parallel Applications. EECE 496 Project Report, 21 (2007)
9. Huang, W.: High Performance Network I/O. In: Virtual Machines Over Modern Interconnects. PhD thesis, The Ohio State University (2008)
10. NASA: NAS parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/ (Last visit: 26-06-2009)
11. Pratt, I., Fraser, K., Spector, S., Guyader, J.: XEN web page, http://www.xen.org (Last visit: 26-06-2009)
12. Borges, G., David, M., Gomes, J., López, J., Rey, P., Simon, A., Fernández, C., Kante, D., Sephton, K.M.: Sun Grid Engine, a New Scheduler for EGEE Middleware. In: IBERGRID'07 (2007)
13. Intel: Intel Linpack, http://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download/ (Last visit: 26-06-2009)
14. Coker, R., Coker, F.: Bonnie++ webpage, http://www.coker.com.au/bonnie++/ (Last visit: 26-06-2009)
15. Norcott, W., Capps, D.: Iozone, http://www.iozone.org/ (Last visit: 26-06-2009)
16. Gates, M., Warshavsky, A.: Iperf, http://sourceforge.net/projects/iperf/ (Last visit: 26-06-2009)
17. Rabenseifner, R., Schulz, G.: Effective Bandwidth Benchmark, https://fs.hlrs.de//projects/par/mpi/b_eff/ (Last visit: 26-06-2009)
18. CESGA: Complete benchmarks results, http://www3.egee.cesga.es/VHPC09/results.tgz (Last visit: 02-10-2009)
19. HP: SFS Web page, http://h20341.www2.hp.com/HPC/cache/276636-0-0-0-121.html (Last visit: 26-06-2009)
20. Frisch, M., Hess, J.: Gaussian webpage, http://www.gaussian.com/ (Last visit: 26-06-2009)
21. van der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A.E., Berendsen, H.J.C.: Gromacs: Fast, flexible and free. J. Comp. Chem. 26, 1701–1718 (2005)

Real-Time Issues in Live Migration of Virtual Machines[*]

Fabio Checconi[1], Tommaso Cucinotta[1], and Manuel Stein[2]

[1] Scuola Superiore Sant'Anna, Pisa, Italy
[2] Alcatel-Lucent Bell Labs, Stuttgart, Germany

Abstract. This paper addresses the issue of how to meet the strict timing constraints of (soft) real-time virtualized applications while the Virtual Machine (VM) hosting them is undergoing a live migration. To this purpose, it is essential that the resource requirements of a migration are identified in advance, that appropriate resources are reserved to the process, and that multiple VMs sharing the same resources are temporally isolated from each other. The first issue is dealt with by introducing a stochastic model for the migration process. The other ones by introducing a methodology making use of proper scheduling algorithms (for both CPU and network) that allow for reserving resource shares to individual VMs. Also, an extensive set of simulations have been done by using traces of a VLC video server virtualized by using KVM on Linux. The traces have been obtained by patching KVM at the kernel level, and the same patch constitutes an important step towards the complete implementation of the proposed technique. The obtained results highlight the benefits of the proposed approach.

1 Introduction

Virtualization technology is gaining more and more interest in the high-performance computing world. However, its use implies a level of sharing of the available hardware resources that is unprecedented. In fact, now it is possible to share the same physical host (PH) across multiple concurrently running OS instances, and it is possible to live-migrate an entire OS to a different PH if needed, with very limited service interruption times. Unfortunately applications may suffer from the concurrent access to shared resources, like CPUs, network links and storage, if proper allocation policies are not adopted. This is of particular relevance whenever the hosted applications exhibit real-time/QoS requirements. This kind of constraints is in place not only for interactive applications but also for batch activities whose QoS levels need to adhere to precise service-level agreements.

One key issue about guaranteeing proper QoS levels is how to keep control of what exactly happens during the live migration of a VM. For example, the

[*] The research leading to these results has been supported by the European Commission under grant agreement n.214777, in the context of the IRMOS Project. More information at: `http://www.irmosproject.eu`

time needed to migrate a VM from one PH to another may be highly variable depending on the network load that is being generated by other VMs on the subnet, as well as on the computational load that is being generated by other VMs concurrently running on the migration start and end PHs.

Contributions of This Paper. This paper addresses the issue of how to guarantee that the process of VM live migration exhibits a temporal behavior that may be kept under control. This problem is faced with by investigating the resource requirements needed by the migration process, and by proposing mechanisms that may be used to guarantee appropriate resource availability when the live migration occurs. To this purpose, a theoretical framework is introduced for estimating the variability of the duration of the overall migration time and of the down-time, with respect to the page migration policy. Also, the issue of how to guarantee that the migration process itself does not interfere too much with the other running VMs is addressed.

2 Background

Migration is the process used to transfer a VM from the host on which it resides to a different one. A migration is said to be *live* if the execution of the VM is not interrupted during most of the transfer.

As described in [1], when a Virtual Machine Monitor (VMM) has to migrate a VM to a new PH, all the resources associated with the VM have to be transferred to the new host, comprising the memory, the internal state of the devices and of the virtual CPU. The most time-consuming resource to transfer is generally the memory. in several ways, and This paper focuses on the so-called *pre-copy* of memory pages, where the VMM begins transferring the pages while the VM is running, checks what pages are dirtied again after the transfer, and retransmits them. This process is repeated for a certain number of times, after which the VM is stopped, the remaining dirty pages are transmitted and the VM is started on the destination.

In these systems, migration of storage is not usually a concern, because of the use of network-based storage solutions that allow for a client VM to seamlessly keep accessing the data after migration to a different host.

Related Work. The use of migration in virtualized environments is a well known subject in the literature. The first proposed mechanisms just stopped VMs on the source PH, saving their state, and restarting them using the saved state on the remote PH (see, e.g., [2]). These approaches suffer of long down times, that often are not acceptable when VMs are executing interactive applications, and furthermore make it impossible to keep the VM connections active.

To overcome these limitations, live migration has been introduced, where the transfer of the VM memory is done while the VM is running. *Pre-copy* migration was introduced in [3], the VM is restarted at and later used in NomadBIOS [4], a VMM built on top of the L4 microkernel [5], and then in Xen [6].

With *demand migration*, derived from the *copy-on-reference* mechanism described in [7], memory is migrated to the destination after the VM has restarted its execution remotely. Recently, in [8], the authors introduced *post-copy* migration, which works enhancing the demand-migration approach by reordering the page transfers for the purpose of minimizing the time spent by the VM waiting for transfers after it restarts execution. With *self-migration*, introduced in [9], the guest OS has awareness of being executed inside a VM, and exploits standard checkpointing techniques to transfer the memory and internal state of the OS by itself.

In [10], a comparison between classical static VM allocation and dynamic resource reallocation enabled by live migration is done. The authors of [11] introduced an analytical model of VM migration to estimate the expected improvement of the hosted service's response time due to a migration decision.

The issues of temporal isolation across multiple VMs concurrently running on the same host, and of how to run real-time virtualized tasks under such conditions, have been considered in [12] and [13] by (partially) the same authors of this paper, but live-migration has not been addressed yet.

3 Live Migration Model

This section presents a stochastic model for real-time migration, whose purpose is twofold: on one hand, it allows for identifying the requirements of the live migration process; on the other hand, it constitutes a motivation for the ordering of pages that are used in the process, as it will be detailed in Section 5.

A VM is characterized by a set of memory pages $\{p_1, \ldots, p_N\}$ assumed for simplicity to be of fixed size equal to P bytes. Assume the bandwidth available for the transfer is constant and equal to b bytes per second (this is possible by using the techniques described in Section 4), and let T denote the time interval needed to transfer a single page $T = \frac{P+H}{b}$ (under the assumption that no compression is used), where H is the overhead in bytes introduced by the migration protocol for each page. Assume that, for the time horizon spanning the entire migration process, each page p_i has a constant probability π_i of being accessed at least once for writing within each time frame T, and assume the events of write access for each page are all independent from one another. Assume the migration process works according to the following steps:

1. at time t_1 in which the migration starts, the set of pages $\mathcal{D}_1$ to be transmitted is set to the entire set of pages used by the VM; let n_1 denote its cardinality $n_1 = |\mathcal{D}_1|$;
2. for $k = 1, \ldots, K$, repeat the following: all the n_k pages in $\mathcal{D}_k$ ($n_k = |\mathcal{D}_k|$) are transferred, with a bandwidth of b bytes per second, according to the order specified by the function $\phi_k : \{1 \ldots n_k\} \to \{1 \ldots N\}$ (i.e., the pages are transmitted in the order $p_{\phi_k(1)}, \ldots, p_{\phi_k(n_k)}$); the transfer ends at $t_{k+1} = t_k + n_k T$, in which n_{k+1} pages $\mathcal{D}_{k+1}$ are found to have become dirty again;
3. stop the VM and transfer the last n_{K+1} pages in $\mathcal{D}_{K+1}$, up to the migration finishing time $t_f = t_{K+1} + n_{K+1}\frac{P+H}{b_d}$, using a bandwidth of b_d bytes per second, with $b_d \geq b$.

Then, the crucial values characterizing the migration process are the down-time $t_d = t_f - t_K$ during which the VM is stopped, and the overall migration time $t_{tot} = t_f - t_1$, which may now be expressed in terms of the other quantities introduced above:

$$t_d = \left(\frac{P+H}{b_d}\right) n_{K+1}, \quad t_{tot} = \left(\frac{P+H}{b}\right) \sum_{k=1}^{K} n_k + t_d. \tag{1}$$

The above introduced notation and assumptions are at the basis of the following results, that focus on the case $K = 1$ for the sake of brevity. All proofs are omitted but they are available at: `http://retis.sssup.it/~tommaso/vhpc09-proofs.pdf`

Proposition 1. *The probability of a page p_i that is not dirty at time t_1 to become dirty and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \notin \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1}. \tag{2}$$

Proposition 2. *The probability of a page p_i that is dirty at time t_1 to become dirty again and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \in \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1 + 1 - \phi_1^{-1}(i)}, \tag{3}$$

where $\phi_1^{-1}(\cdot) : \{1 \ldots N\} \to \{1 \ldots n_1\}$ denotes the inverse of the $\phi_1(\cdot)$ function.

Theorem 1. *The expected overall migration time (with $K = 1$) is:*

$$E[t_{tot}] = \left(\frac{P+H}{b}\right) n_1 + \left(\frac{P+H}{b_d}\right) \left[n_1 - \sum_{j=1}^{n_1} \left(1 - \pi_{\phi_1(j)}\right)^{n_1 + 1 - j} + (N - n_1) - \sum_{i \notin \mathcal{D}_1} (1 - \pi_i)^{n_1} \right]. \tag{4}$$

Theorem 2. *The order $(\phi_k(1), \ldots, \phi_k(n_k))$ of transmission of the pages that minimizes the expected number of dirty pages found at the end of the k^{th} live migration step must satisfy the following condition:*

$$\forall j \; \pi_{\phi_k(j)} (1 - \pi_{\phi_k(j)})^{n_k - j} \leq \pi_{\phi_k(j+1)} (1 - \pi_{\phi_k(j+1)})^{n_k - j}. \tag{5}$$

Corollary 1. *If the probabilities π_i are all lower than $\frac{1}{n_k+1}$, than the optimum ordering is obtained for increasing values of the probabilities π_i. On the other hand, if the probabilities are all greater than $\frac{1}{2}$, then the optimum ordering is obtained for decreasing values of the π_i.*

Reducing the Overall Migration Time. Among the pages to be transmitted there are pages which are accessed by the VM with a very high frequency. For example, pages containing data used by the guest OS for scheduling processes are found as always written at each observation instant. Therefore, it is not convenient to transmit such pages at each migration step, because these pages would need to be transmitted again in the last step, when the VM is stopped. Therefore, it is possible to modify the migration algorithm as follows: among the n_k pages that are found as dirty at start of step k, for a set of pages $F_k \subset \mathcal{D}_k$ delay the transmission to when the VM is stopped. One possibility is to choose the pages for which the access probabilities are higher than a threshold value $\overline{\pi}$: $F_k \triangleq \{p_i \in \mathcal{D}_k \mid \pi_i \geq \overline{\pi}\}$. Focusing on $K = 1$, the following holds:

Proposition 3. *If the transmission of the pages* $F_1 \triangleq \{p_i \in \mathcal{D}_1 \mid \pi_i \geq \overline{\pi}\}$ *is delayed, then the new overall transmission time* $\tilde{t}_{tot}$ *and the new down-time* $\tilde{t}_d$ *satisfy the following:*

$$E\left[\tilde{t}_d\right] \leq E\left[t_d\right] + |F_1|\left(1 - \overline{\pi}\right)\left(\frac{P+H}{b_d}\right)$$

$$E\left[\tilde{t}_{tot}\right] \leq E\left[t_{tot}\right] - |F_1|\left(\frac{P+H}{b}\right) + |F_1|\left(1 - \overline{\pi}\right)\left(\frac{P+H}{b_d}\right).$$

Therefore, with a sufficiently low $\overline{\pi}$, it is possible to achieve a negligible increase in the expected down-time but with a substantial decrease of the overall migration duration. Simulations shown in Section 5 will confirm this.

Practical Implications. The theoretical framework introduced above relies on precise knowledge of the probability π_i of access of a VM to each page p_i. From a practical perspective, these probabilities are not actually known, however it is possible to estimate their values at run-time. One possible way of doing this is by sampling periodically, with a fixed period what pages in the VM have been tagged as dirty by the kernel, then resetting the dirty flag bits and repeating the measure several times.

Building an accurate ordering of dirty pages satisfying Equation 5 may lead to non-negligible overheads (actually, a possible algorithm for doing this is still being developed). Therefore, this paper proposes a couple of algorithms: a simple LRU based approach, where the pages that are transmitted first during each live migration step are the least recently used ones, and a frequency-based approach, where the pages are transmitted in order of increasing access frequency. The results section will show results obtained with the introduced orderings.

4 Real-Time Issues

In order to keep control over the live migration process, so as to achieve a predictable timing behavior of the process, the technique proposed in this paper foresees the adoption of proper scheduling strategies for the involved resources, namely computation and network resources.

Computation Resources. In order to guarantee proper processor shares to Virtual Machines that are concurrently running on the same Physical Host, it is possible to exploit scheduling strategies that are available for the Linux kernel as separate patches in the domain of soft real-time applications.

For example, the AQuoSA scheduler [14] developed in the context of the FRESCOR[1] European Project provides a user-space API and a set of command-line tools that allow for providing to a thread, process or group of them, a scheduling guarantee by the kernel. Such guarantee is expressed in terms of a pair (Q, P), with the meaning that within each period of duration P microseconds, the thread (or thread group) is reserved the CPU for Q microseconds. This guarantee has a strong theoretical foundation, as the scheduler has been written as a variation of the Constant Bandwidth Server, a real-time scheduler whose description can be found in [15].

Also the POSIX Sporadic Server [16] and the framework presented in [17], both recently developed in the context of the IRMOS European Project[2] may be used for the purpose of temporal isolation among multiple processes running on the same Linux OS.

Such mechanisms may be used in order to encapsulate each VM (along with all the threads it is composed of) within a proper scheduling guarantee, whose parameters need to be found by using appropriate benchmarking techniques.

Network Resources. For the estimation of the overall number of page transmissions during the live migration to be accurate, the network bandwidth b should be of a constant available bitrate. With a fixed size of P bytes per page, the time slot T for a page to become dirty has a hyperbolic dependence on the bandwidth b and so is any variation ΔT. Note, that the average probability π_i of a page being accessed in a single slot T increases if the network bandwidth decreases. Further, the recursive nature of the iterative process accounts for an autocorrelated contribution of T to the overall number of required page retransmissions during each iteration.

Practically, the longer a page transmission takes, the higher is the probability of pages being dirtied which increases the number of required retransmissions. With more pages being dirtied, the duration of the follow-up iteration increases during which more pages can be dirtied again. ΔT propagates analogously, because a single variation of the time required for a page transmission results in an autocorrelated propagation of error in the estimation model.

In a managed network computing cluster, the available bandwidth for a possible migration can be properly reserved. An example for an innovative infrastructure that considers network resource isolation and reservation under multi-hop conditions for cloud computing is the Intelligent Service-Oriented Network Infrastructure (ISONI) of the IRMOS project. For the assumptions of a constant duration T per page transmission to hold, a constant bitrate scheduling algorithm is recommended to determine whether the required overall migration time and expected VM downtime are acceptable for the efficiency of the infrastructure

[1] More information at the URL: `http://www.frescor.org`

[2] More information at the URL: `http://www.irmosproject.eu`

and the SLA of the VM, respectively. Further research would be required to account for the effect and the allowable degree of bandwidth variations during the migration mechanism.

5 Evaluation Results

Implementation. In order to support the algorithms described in this paper, the basic infrastructure has been implemented modifying the KVM hypervisor and the Linux kernel itself. This comprises a page tracing mechanism [3] that exposes to user-space the set of pages that have been accessed in write (dirtied) within each observation interval. Page accesses are traced using a bitmap inside the hypervisor; every time a writable mapping is created by the guest, the hypervisor sets the bitmap position corresponding to the newly mapped page. Periodically the bitmap is zeroed and all the writable mappings are reset to read-only. Currently, a simple user-space program saves this information to trace files, which have been used for the simulations shown later. A complete implementation of the live-migration mechanisms proposed in this paper would just require to exploit this information to affect the transmission order of the pages in the existing KVM migration code.

Simulations. This section reports simulation results that prove effectiveness of the proposed technique in reducing both the overall migration time and the down-time. Simulations rely upon traces gathered from real virtualized applications running on KVM on Linux, patched with the tracer previously described in this section.

A virtualized VideoLAN Client (VLC) server has been chosen as the target application scenario for the evaluation of the proposed technique. This scenario raises interesting real-time issues (concerning the down time during a live migration) in the case of live streaming of a video acquired from a camera and transmitted in real-time to the viewers, or in the case of a video streaming service that needs a low seeking latency (as needed during an interactive distributed video editing session). A few traces have been collected independently on VMs that were running the just cited application, while a few clients were accessing the provided service. The observation period has been set to $250ms$. In the collected trace, the VM had a set of about 6500 mapped pages (with 16 KBytes per page) when the migration was simulated. This would correspond to an service interruption of about 8 seconds, if a stop-and-transfer were performed with a bandwidth of 100 MBit/sec (plus the protocol overheads) reserved to the process.

Figure 1 shows the results obtained by simulating the live migration process, in terms of achieved number of down-pages, transmitted while the VM is stopped, and overall number of transmitted pages during the entire migration. These results have been measured for various number of live migration steps/rounds, which correspond to the different points on each curve. Also, multiple page transmission policies have been used in the simulation of the same

[3] The tracing patch used to collect the data used in the experimental section is available at `http://feanor.sssup.it/~fabio/linux/kvm/page-trace/`

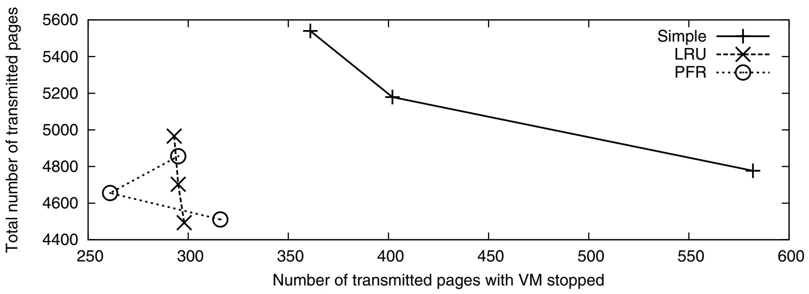


Fig. 1. Performance achieved at varying values of K (various points within each curve) and page migration policy (various curves)

live migrations, corresponding to the different curves on the figure. The `Simple` curve refers to the standard address-based ordering of pages, which is used by default by the current version of KVM, the `LRU` curve is obtained when transmitting first the Least Recently Used pages, and finally the `PFR` curve is obtained when transmitting first the pages which have been dynamically measured (at run-time) to have the lowest write access frequency. The bandwidth guaranteed to the migration process has been set to 50 Mbit/s.

The rightmost points correspond to $K = 1$ and the leftmost points correspond to $K = 3$. Increasing K, generally leads to an increase of the overall transmission time, but also to a significant reduction in the down-time. For example, with the standard KVM policy, with a single round there are 400 pages left to transmit when the VM is stopped, whilst with 3 rounds these pages are reduced to about 270 (roughly a 48% down-time reduction).

Comparing the rightmost, leftmost and middle points on each curve, it is evident how a simple LRU reordering of the pages transmitted during a live migration, as compared to the default address-based order, can achieve a great reduction in both the down-time and the overall migration time: the pages left to transmit when the VM is stopped can decrease from about 570 down to about 300 (47% down-time reduction) when $K = 1$, or a decrease from about 360 to about 290 (19.4% reduction) with $K = 3$. Also, the overall number of pages to transmit during the migration is reduced from about 4800 down to 4500 (6.25% reduction) with $K = 1$, or from about 5500 down to about 5000 (9.1% reduction).

The figure also highlights that a further down-time decrease may be obtained by transmitting the pages in increasing order of write access probability (PFR curve), however its relevance needs to be compared with the additional overhead of keeping an exact ordering by frequency of access, as compared to the simple LRU ordering, which may be implemented by a list that is manipulated in $O(1)$.

Figure 2 reports a similar plot obtained when an LRU page transmission policy has been used, with the additional trick to delay transmission of the most frequently accessed pages. The various curves in the figure correspond to different values of the threshold value $\overline{\pi}$ (indicated in the legend) for the page access probability, over which the page transmission has been delayed (as described in Section 3). The further improvements achieved by this technique

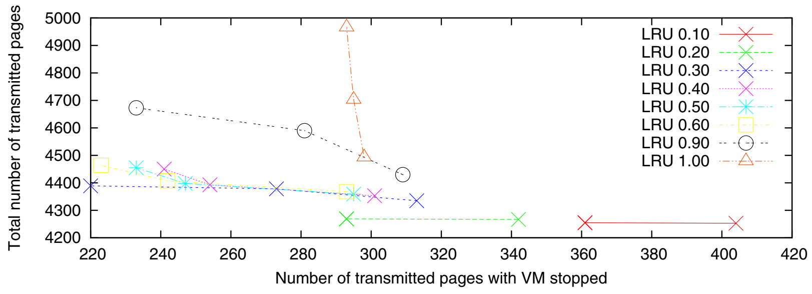


Fig. 2. Performance achieved at varying values of K (various points within each curve) and of $\overline{\pi}$ (various curves)

are evident from the figure, where the down-pages may be further reduced from 300 down to 220 (27% reduction), and the overall time from 5000 down to 4400 (12% decrease) when using a $\overline{\pi} = 0.30$.

6 Conclusions

This paper presented a technique for live migration of real-time virtualized applications. The achievement of very low and predictable down times, as well as the availability of guaranteed resources during the migration process, are key factors for obtaining outages in the provided service with a negligible impact.

A probabilistic model of the migration process has been introduced, for the purpose of building a sound mathematical theory over which to found a novel set of migration policies, such as the proposed one based on a simple LRU order, or the more complex one based on the observed page access frequencies in the past VM history. The LRU policy has been proved (by simulation) to achieve a good degree of effectiveness in decreasing the down-time and overall migration time, still keeping an acceptable level of overhead.

However, the proposed technique needs a deeper evaluation over the full implementation that is being developed, especially on the side of the possible trade-offs between accuracy in gathering the information needed for optimizing the page transmission order, the corresponding run-time overhead, and possible variations to the page transmission schemes adopted.

References

1. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: NSDI'05: Proceedings of the 2nd Symposium on Networked Systems Design & Implementation, Berkeley, CA, USA, pp. 273–286. USENIX Association (2005)
2. Kozuch, M., Satyanarayanan, M.: Internet suspend/resume. In: WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, Washington, DC, USA, p. 40. IEEE Computer Society, Los Alamitos (2002)

3. Theimer, M.M., Lantz, K.A., Cheriton, D.R.: Preemptable remote execution facilities for the v-system. SIGOPS Oper. Syst. Rev. 19(5), 2–12 (1985)
4. Hansen, J.G., Henriksen, A.K.: Nomadic operating systems. Master's thesis, Dept. of Computer Science, University of Copenhagen, Denmark (2002)
5. Härtig, H., Hohmuth, M., Liedtke, J., Shönberg, S., Wolter, J.: The performance of micro-kernel-based systems. In: SOSP '97: Proceedings of the Sixteenth ACM Symposium on Operating System Principles, pp. 66–77. ACM, New York (1997)
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 164–177. ACM, New York (2003)
7. Zayas, E.: Attacking the process migration bottleneck. SIGOPS Oper. Syst. Rev. 21(5), 13–24 (1987)
8. Hines, M.R., Gopalan, K.: Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In: VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pp. 51–60. ACM, New York (2009)
9. Hansen, J.G., Jul, E.: Self-migration of operating systems. In: EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop, p. 23. ACM, New York (2004)
10. Kochut, A.: On impact of dynamic virtual machine reallocation on data center efficiency. In: MASCOTS '08: Proceedings of the 2008 16th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, September 2008, pp. 1–8 (2008)
11. Kochut, A., Beaty, K.: On strategies for dynamic resource management in virtualized server environments. In: MASCOTS '07: Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Washington, DC, USA, pp. 193–200. IEEE Computer Society, Los Alamitos (2007)
12. Cucinotta, T., Anastasi, G., Abeni, L.: Real-time virtual machines. In: Proceedings of the 29th IEEE Real-Time System Symposium (RTSS 2008) – Work in Progress Session, Barcelona (December 2008)
13. Cucinotta, T., Anastasi, G., Abeni, L.: Respecting temporal constraints in virtualised services. To appear in Proceedings of the 2nd IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications (RTSOAA 2009), Seattle, Washington (July 2009)
14. Palopoli, L., Cucinotta, T., Marzario, L., Lipari, G.: AQuoSA — adaptive quality of service architecture. Software – Practice and Experience 39(1), 1–31 (2009)
15. Abeni, L., Buttazzo, G.: Integrating multimedia applications in hard real-time systems. In: Proceedings of the IEEE Real-Time Systems Symposium, Madrid, Spain (December 1998)
16. Faggioli, D., Mancina, A., Checconi, F., Lipari, G.: Design and implementation of a POSIX compliant sporadic server. In: Proceedings of the 10th Real-Time Linux Workshop (RTLW), Mexico (October 2008)
17. Checconi, F., Cucinotta, T., Faggioli, D., Lipari, G.: Hierarchical multiprocessor cpu reservations for the linux kernel. In: Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009), Dublin, Republic of Ireland (July 2009)

Appendix: Proofs

Proposition (Was Proposition 1 in the text). *The probability of a page p_i that is not dirty at time t_1 to become dirty and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \notin \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1} . \tag{6}$$

Proof. The probability may be written as the complement of the probability of the page never being accessed during the transmission of the n_1 pages. Let $E_{i,\,j}$ denote the probability that p_i is not accessed during the transmission of the j^{th} page, with $j = 1, \ldots, n_1$. Then, due to the assumptions on the page access probabilities, the $\{E_{i,\,j}\}_{j=1,\ldots,n_1}$ events are all independent, therefore: $\Pr\{p_i \in \mathcal{D}_2 \mid p_i \notin \mathcal{D}_1\} = 1 - \Pr\{E_{i,\,1} \wedge \ldots \wedge E_{i,\,n_1}\} = 1 - \prod_{j=1}^{n_1} \Pr\{E_{i,\,j}\} = 1 - (1 - \pi_i)^{n_1}$. □

Proposition (Was Proposition 2 in the text). *The probability of a page p_i that is dirty at time t_1 to become dirty again and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \in \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1 + 1 - \phi_1^{-1}(i)} , \tag{7}$$

where $\phi_1^{-1}(\cdot) : \{1 \ldots N\} \to \{1 \ldots n_1\}$ denotes the inverse of the $\phi_1(\cdot)$ function.

Proof. The probability may be written as the complement of the probability of the page never being accessed in the period going from the time the page starts to be transmitted $t_1 + \phi_1^{-1}(i)T$, to the time the transmission round is over $t_1 + n_1 T$, corresponding to a total of $n_1 + 1 - \phi_1^{-1}(i)$ time slots of duration T, i.e., $\Pr\{p_i \in \mathcal{D}_2 \mid p_i \in \mathcal{D}_1\} = 1 - \Pr\left\{E_{i,\,\phi_1^{-1}(i)} \wedge \ldots \wedge E_{i,\,n_1}\right\} = 1 - \prod_{j=\phi_1^{-1}(i)}^{n_1} \Pr\{E_{i,\,j}\} = 1 - (1 - \pi_i)^{n_1 + 1 - \phi_1^{-1}(i)}$. □

Theorem (Was Theorem 1 in the text). *The expected overall migration time (with $K = 1$) is:*

$$E\left[t_{tot}\right] = \left(\frac{P+H}{b}\right) n_1 + \left(\frac{P+H}{b_d}\right) \left[n_1 - \sum_{i \in \mathcal{D}_1} (1 - \pi_1)^{n_1 + 1 - \phi_1^{-1}(i)} \right. \\ \left. + (N - n_1) - \sum_{i \notin \mathcal{D}_1} (1 - \pi_i)^{n_1} \right] . \tag{8}$$

Proof. From Equation 1, $t_{tot} = \left(\frac{P+H}{b}\right) n_1 + t_d$, where the first term is constant and known, and the expected down-time $E\left[t_d\right]$ may be computed as follows. Let X_i be a stochastic variable equal to 1 if the page p_i is dirty and needs to be transmitted in the next step, and 0 otherwise. Clearly, the number of pages n_2 that are dirty and thus need to be transmitted in the next step, is equal to: $n_2 = \sum_{i=1}^{N} X_i$. However, for each page $i \in \mathcal{D}_1$, the probability that $X_i = 1$ is the probability that the page becomes dirty again in the time-interval $\left[t_1 + \phi_1^{-1}(i)T,\, t_1 + n_1 T\right]$, which is computed by means of Equation 7. On the other hand, for each page $i \notin \mathcal{D}_1$, the probability that $X_i = 1$ is the probability that the page becomes dirty in the time-interval $[t_1,\, t_1 + n_1 T]$, computed by using Equation 6. Therefore:

$$E[n_2] = E\left[\sum_{i=1}^{N} X_i\right] = \sum_{i\in\mathcal{D}_1} E[X_i] + \sum_{i\notin\mathcal{D}_1} E[X_i] \tag{9}$$

$$= \sum_{i\in\mathcal{D}_1}\left[1 - (1-\pi_i)^{n_1+1-\phi_1^{-1}(i)}\right] + \sum_{i\notin\mathcal{D}_1}\left[1-(1-\pi_i)^{n_1}\right] \tag{10}$$

$$= n_1 - \sum_{i\in\mathcal{D}_1}(1-\pi_i)^{n_1+1-\phi_1^{-1}(i)} + (N-n_1) - \sum_{i\notin\mathcal{D}_1}(1-\pi_i)^{n_1} \tag{11}$$

The proof is easily obtained by considering that $E[t_d] = \left(\frac{P+H}{b_d}\right) E[n_2]$. □

Theorem (Was Theorem 2 in the text). *The order $(\phi_k(1),\ldots,\phi_k(n_k))$ of transmission of the pages that minimizes the expected number of dirty pages found at the end of the k^{th} live migration step must satisfy the following condition:*

$$\forall j\ \pi_{\phi_k(j)}(1-\pi_{\phi_k(j)})^{n_k-j} \le \pi_{\phi_k(j+1)}(1-\pi_{\phi_k(j+1)})^{n_k-j}. \tag{12}$$

Proof. During the k^{th} algorithm step, n_k pages are transmitted in the order $(\phi_k(1),\ldots,\phi_k(n_k))$. Therefore, the j^{th} transmitted page $p_{\phi_k(j)}$ has $n_k - j + 1$ time slots for becoming dirty again before the next step. The probability of the page becoming dirty again between the time in which it starts to be transmitted, and the time in which the step finishes, is: $1 - \left(1-\pi_{\phi_k(j)}\right)^{n_k+1-j}$. Now, let X_j be a stochastic variable equal to 1 if the page $p_{\phi_k(j)}$ gets dirty again and needs retransmission in the next step, and 0 otherwise. Clearly, the number N_k of pages that, after being transmitted during step k, become dirty again and need to be retransmitted in the next step, is equal to: $N_k = \sum_{j=1}^{n_k} X_k$. As a consequence, the expected value of N_k may be written as:

$$E[N_k] = \sum_{j=1}^{n_k} E[X_j] = \sum_{j=1}^{n_k} \phi_{\phi_k(j)}(n_k - j + 1) = n_k - \sum_{j=1}^{n_k}\left(1-\pi_{\phi_k(j)}\right)^{n_k-j+1}. \tag{13}$$

At this point, the theorem proof is easily obtained by absurd. Suppose the minimum value of $E[N_k]$ is obtained when the pages do not respect the ordering in 12. Then, there exists an index j such that Condition 12 does not hold. The contribution to $E[N_k]$ due to the two pages is $2 - \left(1-\pi_{\phi_k(j)}\right)^{n_k-j+1} - \left(1-\pi_{\phi_k(j+1)}\right)^{n_k-(j+1)+1}$, while swapping the two pages into the sequence would lead to a contribution of $2 - \left(1-\pi_{\phi_k(j)}\right)^{n_k-(j+1)+1} - \left(1-\pi_{\phi_k(j+1)}\right)^{n_k-j+1}$, leaving the contributions due to the other pages unchanged. This, under the assumption of $\pi_{\phi_k(j)}\left(1-\pi_{\phi_k(j)}\right)^{n_k-j} > \pi_{\phi_k(j+1)}\left(1-\pi_{\phi_k(j+1)}\right)^{n_k-j}$, corresponds to a *decrease* of $E[N_k]$, leading to a contradiction. □

Corollary (Was Corollary 1 in the text). *If the probabilities π_i are all lower than $\frac{1}{n_k+1}$, than the optimum ordering is obtained for increasing values of the probabilities π_i. On the other hand, if the probabilities are all greater than $\frac{1}{2}$, then the optimum ordering is obtained for decreasing values of the π_i.*

Proof. In the first case, for all positive exponents $n_k - j$, the function $\pi(1-\pi)^{n_k-j}$ is always monotonically increasing in π, therefore Condition 12 is equivalent to having increasing probabilities $\pi_{\phi_k(i)}$ in the sequence $\left\{\pi_{\phi_k(1)}, \ldots, \pi_{\phi_k(n_k)}\right\}$. Similarly, in the first case, the function $\pi(1-\pi)^{n_k-j}$ is monotonically decreasing in π. Therefore, the same condition translates to requiring decreasing values of $\pi_{\phi_k(i)}$ in the sequence $\left\{\pi_{\phi_k(1)}, \ldots, \pi_{\phi_k(n_k)}\right\}$. □

Proposition (Was Proposition 3 in the text). *If the transmission of the pages $F_1 \triangleq \{p_i \in \mathcal{D}_1 \mid \pi_i \geq \overline{\pi}\}$ is delayed, then the new overall transmission time $\tilde{t}_{tot}$ and the new down-time $\tilde{t}_d$ satisfy the following:*

$$E\left[\tilde{t}_d\right] \leq E\left[t_d\right] + |F_1|\left(1-\overline{\pi}\right)\left(\frac{P+H}{b_d}\right)$$

$$E\left[\tilde{t}_{tot}\right] \leq E\left[t_{tot}\right] - |F_1|\left(\frac{P+H}{b}\right) + |F_1|\left(1-\overline{\pi}\right)\left(\frac{P+H}{b_d}\right).$$

Proof. If these pages were transmitted, then $E[|F_{k+1}|]$ would be equal to $|F_k| - \sum_{i \in F_k} \overline{\pi}_i^{n_k - j_k(i)+1}$, where $j_k(i)$ is the transmission position of the page i. This may be upper bounded by $E\left[|F_{k+1}|\right] \geq |F_k| - \sum_{i \in F_k}(1-\tilde{\pi}) = |F_k|\,\tilde{\pi}$. Similarly, at the end of the last step (the K^{th}), the number of pages in F_1 that would have to be retransmitted would be $|F_k|\left[1-(1-\tilde{\pi})^K\right]$. So, if these pages are not transmitted at the k^{th} step, then a contribution of $\sum_{k=1}^{K} |F_1|\left[1-(1-\tilde{\pi})^k\right]$ pages is removed from the expected number of pages to be transmitted before stopping the VM, but a contribution of $|F_1|(1-\tilde{\pi})$ would be added to the expected number of pages to be transmitted while the VM is stopped. □

Author Index