Luc Bougé   Martti Forsell
Jesper Larsson Träff   Achim Streit
Wolfgang Ziegler   Michael Alexander
Stephen Childs (Eds.)

Euro-Par 2007 Workshops Parallel Processing

HPPC 2007, UNICORE Summit 2007, and VHPC 2007
Rennes, France, August 28-31, 2007
Revised Selected Papers

Springer

Volume Editors

Luc Bougé
IRISA/ENS Cachan, Rennes, France
E-mail: luc.bouge@bretagne.ens-cachan.fr

Martti Forsell
VTT Technical Research Center of Finland, Oulu
E-mail: martti.forsell@vtt.fi

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

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

Wolfgang Ziegler
Fraunhofer Institute SCAI, Sankt Augustin, Germany
E-mail: wolfgang.ziegler@scai.fraunhofer.de

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

Stephen Childs
Trinity College Dublin, Ireland
E-mail: stephen.childs@cs.tcd.ie

Library of Congress Control Number: 2008921917

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

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

ISSN 0302-9743
ISBN-10 3-540-78472-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-78472-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12236053 06/3180 5 4 3 2 1 0

Preface

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

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

Euro-Par is an annual series of international conferences dedicated to the promotion and advancement of all aspects of parallel and distributed computing. The 2007 event was the 13th issue of the conference. Euro-Par has for a long time been eager to attract colocated events sharing the same goal of promoting the development of parallel and distributed computing, both as an industrial technique and an academic discipline, extending the frontier of both the state of the art and the state of the practice. Since 2006, Euro-Par offers researchers the chance to colocate advanced technical workshops back-to-back with the main conference. This is for a mutual benefit: the workshops can take advantage of all technical and social facilities which are set up for the conference, so that the organizational tasks are kept to a minimal level; the conference can rely on workshops to experiment with specific areas of research which are not yet mature enough, or too specific, to lead to an official, full-fledged topic at the conference.

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

CoreGRID Symposium is the major annual event of the CoreGRID European Research Network on Foundations, Software Infrastructures and Applications for large-scale distributed, Grid and peer-to-peer technologies. It is also an opportunity for a number of CoreGRID Working Groups to organize their regular meetings. The proceedings have been published in a specific volume of the Springer CoreGRID series *Towards Next Generation Grids*, edited by Thierry Priol and Marco Vanneschi.

GECON 2007 is the Fourth International Workshop on Grid Economic and Business Model. Euro-Par was eager to attract an event about this very important aspect of grid computing, which has often been overlooked by scientific researchers of the field. This very successful workshop was organized by Jörn Altmann and Daniel J. Veit. Its proceedings are published in a separate volume of Springer's *Lecture Notes in Computer Science* series, number 4685.

HPPC 2007 is the First Workshop on Highly Parallel Processing on a Chip. With a number of both general and special purpose multi-core processors already on the market, it is foreseeable that new designs with a substantial number of processing cores will emerge to meet demands for extremely high performance, dependability, and controllable power consumption in mobile and embedded devices, and in response to the convergence of communication, media and compute devices. This workshop was a unique opportunity for the Euro-Par community to get acquainted with this new and hot field of research.

UNICORE Summit 2007 aimed to bring together researchers and practitioners working with UNICORE in the areas of grid and distributed computing, to exchange and share their experiences, new ideas, and latest research results on all aspects of UNICORE. The UNICORE grid technology provides a seamless, secure, and intuitive access to distributed grid resources. This was the third meeting of the UNICORE community, after a meeting in Sophia-Antipolis, France, in 2005, and a colocated meeting at Euro-Par 2006 in Dresden, Germany, in 2006.

VHPC 2007 is the Workshop on Virtualization/Xen in High-Performance Cluster and Grid Computing. Virtual machine monitors (VMMs) are now integrated with a variety of operating systems and are moving out of research labs into scientific, educational and operational usage. This workshop aimed to bring together researchers and practitioners active in exploring the application of virtualization in distributed and high-performance cluster and grid computing environments. This was a unique opportunity for the Euro-Par community to make connections with this very active research domain.

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

Hosting Euro-Par 2007 and these colocated events in Rennes would not have been possible without the support and the help of different institutions and numerous people.

Although we are thankful to many more people, we are particularly grateful to Édith Blin: she put a huge amount of work in the organization of the conference, always combining efficiency and enthusiasm, smoothing consistently the whole process of organizing the conference.

We are obviously most thankful to the workshop organizers: Martti Forsell and Jesper Larsson Träff for HPPC 2007; Achim Streit and Wolfgang Ziegler for UNICORE Summit 2007; and Michael Alexander and Stephen Childs for VHPC 2007. It has been a pleasure to collaborate with them on this project. We definitely thank them for their interest in our proposal and their trust and availability along the entire preparation process.

Euro-Par 2007 was hosted on the University Campus and we would like to thank the Department of Computer Science (IFSIC) of the University of Rennes 1 for the support and infrastructure. We gratefully acknowledge the great financial and organizational support of INRIA and IRISA as well as the support of our institutional sponsors the University of Rennes 1, the Regional

Council, Rennes Métropole, the local council, the Métivier Foundation, the *Pôle de competitivité Images & Réseaux* and the city of Rennes.

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

It has been a great pleasure to work together on this project in Rennes. We hope that the current proceedings are beneficial for sustainable growth and awareness of parallel and distributed computing concepts in future applications.

November 2007

Luc Bougé
Martti Forsell
Jesper Larsson Träff
Achim Streit
Wolfgang Ziegler
Michael Alexander
Stephen Childs

Organization

Euro-Par Steering Committee

Chair

Christian Lengauer	University of Passau, Germany

Vice-Chair

Luc Bougé	ENS Cachan, France

European Representatives

José Cunha	New University of Lisbon, Portugal
Marco Danelutto	University of Pisa, Italy
Rainer Feldmann	University of Paderborn, Germany
Christos Kaklamanis	Computer Technology Institute, Greece
Paul Kelly	Imperial College, UK
Harald Kosch	University of Passau, Germany
Thomas Ludwig	University of Heidelberg, Germany
Emilio Luque	Universitat Autònoma de Barcelona, Spain
Luc Moreau	University of Southampton, UK
Wolfgang E. Nagel	Technische Universität Dresden, Germany
Rizos Sakellariou	University of Manchester, UK

Non-European Representatives

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

Honorary Members

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

Observers

Anne-Marie Kermarrec	IRISA/INRIA, Rennes, France
Domingo Benítez	University of Las Palmas, Gran Canaria, Spain

Euro-Par 2007 Local Organization

Euro-Par 2007 was organized by the IRISA/INRIA research laboratory in Rennes.

Conference Chairs

Anne-Marie Kermarrec	IRISA/INRIA
Luc Bougé	IRISA/ENS Cachan
Thierry Priol	IRISA/INRIA

General Organization

Édith Blin	IRISA/INRIA

Technical Support

Étienne Rivière, Yann Busnel

Publicity

Gabriel Antoniu

Proceedings

Marin Bertier

Secretariat

Patricia Houée-Barbedet, Violaine Tygréat

CoreGRID Coordination

Païvi Palosaari, Olivia Vasselin

Euro-Par 2007 Workshop Program Committees

Workshop on Highly Parallel Processing on a Chip (HPPC)

Program Chairs

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

Program Committee

Gianfranco Bilardi	University of Padova, Italy
Taisuke Boku	University of Tsukuba, Japan
Martti Forsell	VTT, Finland
Jim Held	Intel, USA
Peter Hofstee	IBM, USA
Ben Juurlink	Technical University of Delft, The Netherlands
Darren Kerbyson	Los Alamos National Laboratory, USA
Lasse Natvig	NTNU, Norway
Kunle Olukotun	Stanford University, USA
Wolfgang Paul	Saarland University, Germany
Andrea Pietracaprina	University of Padova, Italy
Alex Ramirez	Technical University of Catalonia and Barcelona Supercomputing Center, Spain
Peter Sanders	University of Karlsruhe, Germany
Thomas Sterling	Caltech and Louisiana State University, USA
Jesper Larsson Träff	NEC Laboratories Europe, Germany
Uzi Vishkin	University of Maryland, USA

UNICORE Summit

Program Chairs

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

Program Committee

Agnès Ansari	CNRS-IDRIS, France
Rosa Badia	Barcelona Supercomputing Center, Spain
Thomas Fahringer	University of Innsbruck, Austria
Donal Fellows	University of Manchester, UK
Anton Frank	LRZ Munich, Germany
Edgar Gabriel	University of Houston, USA
Alfred Geiger	T-Systems SfR, Germany

Odej Kao	Technical University of Berlin, Germany
Paolo Malfetti	CINECA, Italy
Ralf Ratering	Intel GmbH, Germany
Johannes Reetz	Max-Planck-Institut für Plasmaphysik, RZG, Germany
Mathilde Romberg	University of Ulster, UK
Bernd Schuller	Forschungszentrum Juelich, Germany
David Snelling	Fujitsu Laboratories of Europe, UK
Stefan Wesner	University of Stuttgart, HLRS, Germany
Ramin Yahyapour	University of Dortmund, Germany

Additional Reviewers

Sven van den Berghe
Morris Riedel

Workshops on Virtualization/XEN in HPC Cluster and Grid Computing Environments

Program Chairs

Michael Alexander	WU Vienna, Austria
Stephen Childs	Trinity College, Dublin, Ireland

Program Committee

Jussara Almeida	Federal University of Minas Gerais, Brazil
Padmashree Apparao	Intel Corp., USA
Hassan Barada	Etisalat University College, UAE
Volker Buege	University of Karlsruhe, Germany
Simon Crosby	Xensource, UK
Peter Dinda	Northwestern University, USA
Marc Fiuczynski	Princeton University, USA
Rob Gardner	HP Labs, USA
William Gardner	University of Guelph, Canada
Marcus Hardt	Forschungszentrum Karlsruhe, Germany
Klaus Ita	WU Vienna, Germany
Sverre Jarp	CERN, Switzerland
Krishna Kant	Intel Corporation, USA
Yves Kemp	University of Karlsruhe, Germany
Naoya Maruyama	Tokyo Institute of Technology, Japan
Jean-Marc Menaud	EMN-INRIA, France
José E. Moreira	IBM T.J. Watson Research Center, USA

Sonja Sewera	WU Vienna, Austria
Dan Stanzione	Arizona State University, USA
Peter Strazdins	Australian National University, Australia
Franco Travostino	Nortel, Canada
Andreas Unterkircher	CERN, Switzerland
Geoffroy Vallée	Oak Ridge National Laboratory, USA
Dongyan Xu	Purdue University, USA

Table of Contents

HPPC 2007: Workshop on Highly Parallel Processing on a Chip (Foreword)

Technological developments are bringing parallel computing back into the limelight after some years of absence from the stage of mainstream computing and computer science between the early 1990 and early 2000s. The driving forces behind this return are mainly technological: increasing transistor densities along with hot chips, leaky transistors, and slow wires – coupled with the infeasibility of extracting significantly more ILP at execution time – make it unlikely that the increase in single processor performance can continue the exponential growth that has been sustained over the last 30 years. To satisfy the needs for application performance, major processor manufacturers are instead counting on doubling the number of processor cores per chip every second year, in accordance with the original formulation of Moore's law. We are therefore on the brink of entering a new era of highly parallel processing on a chip. However, many fundamental unresolved hardware and software issues remain that may make the transition slower and more painful than is optimistically expected from many sides. Among the most important issues are convergence on an abstract architecture, programming model, and language to easily and efficiently realize the performance potential inherent in the technological developments.

The *Workshop on Highly Parallel Processing on a Chip* (HPPC) aims to be a forum for discussing such fundamental issues. It is open to all aspects of existing and emerging/envisaged multi-core (by which is meant: many-core) processors with a *significant amount of parallelism*, especially to considerations on novel paradigms and models and the related architectural and linguistic support. To be able to relate to the parallel processing community at large, which we consider essential, the workshop has been organized in conjunction with Euro-Par, the main European (but international) conference on all aspects of parallel processing.

The call for papers for the HPPC workshop was launched early in the year 2007, and by the submission deadline we had received 20 submissions, which were of good quality and generally relevant to the theme of the workshop. The papers were swiftly and expertly reviewed by the Program Committee, most of them receiving four qualified reviews. The Program Chairs thank the Program Committee for the time and expertise they put into the reviewing work, and for getting it all done within the rather strict time limit. A final decision on acceptance was made by the Program Chairs based on the recommendations from the Program Committee. Being a(n extended) half- day event, there was room for accepting only six of the contributions, resulting in an acceptance ratio of about 30%. Five of the six accepted contributions were presented at the workshop (the paper not presented is as a matter of principle not included in these proceedings), together with two forward-looking invited talks by Uzi

Vishkin and Thomas Sterling on realizing a PRAM-on-a-chip vision and societies of cores and their computing culture.

These post-workshop proceedings include the final versions of the presented HPPC papers, taking the feedback from reviewers and workshop audience into account. In addition, the extended abstracts of the two invited talks by Uzi Vishkin and Thomas Sterling have also been included in the proceedings.

The Program Chairs sincerely thank the Euro-Par organization for providing the opportunity to arrange the HPPC workshop in conjunction with the Euro-Par 2007 conference. We also warmly thank our sponsors VTT and Euro-Par for financial support, which made it possible to invite Uzi Vishkin and Thomas Sterling, both of whom we also sincerely thank for accepting our invitation to come and speak. Finally, we thank all attendees at the workshop, who contributed to a lively day, and hope they too found something of interest in the workshop. Based on the mostly positive feedback, the Program Chairs and organizers plan to continue the HPPC workshop in conjunction with Euro-Par 2008.

November 2007

Martti Forsell
Jesper Larsson Träff

Toward Realizing a PRAM-on-a-Chip Vision

Uzi Vishkin

University of Maryland
University of Maryland Institute for Advanced Computer Studies (UMIACS)
College Park, Maryland, USA

Abstract

Serial computing has become largely irrelevant for growth in computing performance at around 2003. Having already concluded that to maintain past performance growth rates, general-purpose computing must be overhauled to incorporate parallel computing at all levels of a computer system – including the programming mode – all processor vendors put forward many-core roadmaps. They all expect exponential increase in the number of cores over at least a decade. This welcome development is also a cause for apprehension. The whole world of computing is now facing the same general-purpose parallel computing challenge that eluded computer science for so many years and the clock is ticking. It is becoming common knowledge that if you want your program to run faster you will have to program for parallelism, but the vendors who set up the rules have not yet provided clear and effective means (e.g., programming models and languages) for doing that. How can application software vendors be expected to make a large investment in new software developments, when they know that in a few years they are likely to have a whole new set of options for getting much better performance?! Namely, we are already in a problematic transition stage that slows down performance growth, and may cause a recession if it lasts too long. Unfortunately, some industry leaders are already predicting that the transition period can last a full decade.

The PRAM-On-Chip project started at UMD in 1997 foreseeing this challenge and opportunity. Building on PRAM – a parallel algorithmic approach that has never been seriously challenged on ease of thinking, or wealth of its knowledge-base – a comprehensive and coherent platform for on-chip general-purpose parallel computing has been developed and prototyped. Optimizing single-task completion time, the platform accounts for application programming (VHDL/Verilog, OpenGL, MATLAB, etc), parallel algorithms, parallel programming, compiling, architecture and deep-submicron implementation, as well as backward compatibility on serial code. The approach goes after any type of application parallelism regardless of its amount, regularity, or grain size. Some prototyping highlights include: an eXplicit Multi-Threaded (XMT) architecture, a new 64-processor, 75MHz XMT (FPGA-based) computer, 90nm ASIC tape-out of the key interconnection network component, a basic compiler, class tested programming methodology where students are taught only parallel algorithms and pick the rest on their own, and up to 100X speedups on applications.

The talk will overview some future plans and will argue that the PRAM-On-Chip approach is a promising candidate for providing the processor-of-the-future. It will also posit that focusing on a small number of promising approaches, such as PRAM-On-Chip, and accelerate their incubation and testing stage, would be most beneficial both: (i) for the field as a whole, and (ii) for an individual researcher who is seeking improved impact.
URL: `http://www.umiacs.umd.edu/~vishkin/XMT`

Societies of Cores and Their Computing Culture

Thomas Sterling

Louisiana State University
Center for Computation & Technology
Baton Rouge, Louisiana, USA

Abstract

The performance opportunities enabled through multi-core chips and the efficiency potential of heterogeneous ISA and structures is creating a climate for computer architecture, highly parallel processing chips, and HPC systems unprecedented for more than a decade. But with change comes the uncertainty from competition of alternatives. One thing is clear: all systems will be parallel systems and all chips will be highly parallel. If so, then, how will the parallelism be represented and controlled and what will be the roles and responsibilities for managing system wide parallelism? This presentation will address both the exciting opportunities and challenges of highly parallel processing cores on chips and describe one possible path for future parallel ISA cores, ParalleX, which may enable the synthesis of many cores into one single scalable system.

Hardware Transactional Memory with Operating System Support, HTMOS

Sasa Tomic, Adrian Cristal, Osman Unsal, and Mateo Valero

Barcelona Supercomputing Center
Universitat Politècnica de Catalunya

Abstract. Hardware Transactional Memory (HTM) gives software developers the opportunity to write parallel programs more easily compared to any previous programming method, and yields better performance than most previous lock-based synchronizations.

Current implementations of HTM perform very well with small transactions. But when a transaction overflows the cache, these implementations either abort the transaction as unsuitable for HTM, and let software takeover, or revert to some much more inefficient hash-like in-memory structure, usually located in the userspace.

We present a fast, scalable solution that has virtually no limit on transaction size, has low transactional read and write overhead, works with physical addresses, and doesn't require any changes inside the cache subsystem.

This paper presents an HTMOS - Operating System (OS) and Architecture modifications that leverage the existing OS Virtual Memory mechanisms, to support unbounded transaction sizes, and provide transaction execution speed that does not decrease when transaction grows.

1 Introduction

1.1 Motivation

Transactional Memory (TM) systems can be subdivided into two flavors: Hardware TM (HTM) and Software TM (STM). HTM systems bound TM implementations to hardware to keep the speculative updated state and as such are fast but suffer from resource limitations. In this work, we propose Hardware Transactional Memory with Operating System support (HTMOS) which is complexity-effective, potentially performs on the same order of magnitude with HTM, and is flexible like STM systems. We present a fast, scalable solution that has virtually no limit on transaction size, does not prefer either directory based coherence or snooping, that has low transactional read and write overhead, that works with physical addresses, and does not require any changes inside cache subsystem. Instead, changes are done on the Operating System (OS) level - in the Virtual Memory system, and inside the processor - the TLB and in the form of additional instructions/functionality.

HTMOS involves modest architectural and OS changes to keep the additional copies of a page in memory for transactional memory. Compared to the previous HTM proposals, HTMOS has three important advantages: (1) implicitly accommodates large transactions without getting bogged down with complex implementations which decreases the performance (as is the case for most HTMs). In fact for a 4GB address space, a transaction could be as large as 2GB. (2) is much more flexible than other HTMs. For example, different versioning schemes such as in-place update versus lazy update are trivial to implement. Most other HTM proposals embed a certain versioning scheme in silicon which makes it very difficult to implement other alternative schemes. (3) ensures strong atomicity. Non-transactional loads and stores do not conflict with transactional ones. More specifically, each non-transactional load or store can be seen like a single-instruction transaction to the transactional ones.

1.2 Previous Work

Almost all current HTM implementations assume that transactions are going to be very small in size. Our assumption is that ordinary programmers will try to use transactions whenever they are not sure if they should use them, and for as big segments of the program as they can.

In the current HTM implementations, there are generally two approaches to the version management: lazy and eager, and two for conflict detection: lazy and eager. Two representatives for these are LogTM[5], from University of Wisconsin, with eager-eager, and Transactional Memory Coherence and Consistency[3], from Stanford University, with lazy-lazy conflict detection and version management. According to some researchers, the overall performance of LogTM is a little bit worse than that of TCC[1].

In a recent proposal, called Unbounded Page based Hardware Transactional Memory[2] (PTM), the conflict detection is done on the level of cache-line sized blocks, and it supports unbounded transaction size. All of the transactional state is indexed by physical pages, and is maintained at the memory controller. PTM's Transaction Access Vector (TAV) double linked list tracks the accesses to a page. One shadow page for every physical page is created, which requires the eager conflict detection. The memory controller is responsible for all conflict detection, updating transactional state, and aborting/committing transactions. On abort or commit, the memory controller updates the TAV and the special summary cache for this transaction. Transactions are nested by flattening. The cost of every miss to the TAV cache increases linearly with the length of the TAV list. The length of TAV lists increases with the number of processors (transactions) accessing different blocks of the same physical page. So, if there are many (N) processors, each non-cached memory access would require N memory reads. To avoid the high cost of non-cached access, the TAV cache that was used in evaluation is fully associative 2048 entries, which is very difficult to be done using current technology.

2 HTMOS Architecture

HTMOS leverages the strong coupling between the Architecture and OS to enable the HTM design that is both fast and flexible. The core idea is to create an additional, *secondary* copy of the page, for every transaction, in case when transaction is trying to write to the page. Each transaction has its own transactional *Virtual Page Table* (VPT), which can be utilized to switch between the alternate versions of the block inside the page. Each page is subdivided into blocks of cache-line size, to reduce the granularity for conflict detection. Assuming eager conflict detection implementation, the current value of the data is in the *primary* copy of the page, so only the record-keeping information needs to be cleaned. On abort, the original values are copied from the secondary copy of the page to the primary, and record-keeping information is cleaned. The detailed explanation follows, divided into Software and Hardware sections.

2.1 Software

Operating System manages the memory required for the transactional bookkeeping. It also allocates the secondary pages used for storing the backup copies of the cachelines.

Global Transaction State Table (TST). OS has a special area of memory allocated for the TST. There is a fixed number of transactions running at the same time. This number is fixed by both hardware and software. In our implementation, we use so called transaction flattening: nesting counter is incremented on begin of a nested transaction and decremented on commit of a nested transaction. The real commit is done only when nesting counter reaches zero. Therefore, it is sufficient to have the maximum number of concurrently running transactions equal to number of processors in the system.

The global TST (Tab. 1) holds the basic information about all transactions. Every processor has access to each element of this table and, therefore, can read or set status of any transaction in the system. The table has as many entries as there are processors/transactions in the system. We will assume a 32 processor system.

Each entry of the table is extended with (currently) unused bits up to cacheline size, to minimize the false-sharing between processors.

On system startup, the OS also creates a 32 transactional VPTs. Transactional VPT holds the physical addresses for the secondary pages of this transaction.

Transactional Bitmap (TB). Transactional Bitmap is the key structure of our HTM implementation. This sparse bit-array is permanently stored in the physical memory, and holds the information about the transactional reads and writes from and to the page that it is associated with. It exists *only* for the pages that have the transactional reads and/or writes.

TB is organized in the following way:

For every block in the page (standard page of 4KB is split into 64B standard cacheline-sized blocks) we have a bitmap in the following format: **1 M bit**: marks if this block has been **M**odified by some processor, **32 TXID bits** (1 bit per processor): marks which processors have read the value, or in case when M=1, then which processor (only one) is holding the modified value. This makes a

Table 1. An entry of Transaction State Table, assuming cache-line size of 512b (64B)

size	field name	possible states
52b	tx_vpt	pointer to transactional vpt
1b	active	INACTIVE / INSIDE_PROC
2b	status	RUNNING, COMMITTED, ABORTED, FREE
9b	nesting depth	0..511
64b	ws_size	$0..(2^{64}-1)$
32b	tx_blocked	bitmap of TXs blocked on this one
32b	thread_id	thread_id that is running transaction
320b	unused	-

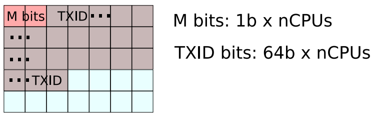


Fig. 1. Transactional Bitmap organization for *nCPUs* processors

total of 64 x (1+32) bits = 2112bits = 264B associated with each transactional PTE, and this is enough to cover up to 32 processors, with concurrent access to each of the 64B blocks in the page.

As can be seen on Fig. 1, when stored in the memory, M bits are grouped into one 64-bit M field, and all TXID bits are grouped in a field after that.

The motivation for the addition of the TB is the *reduction of conflict detection granularity*. The best granularity for the conflict detection is word-size, but false-conflicts are mostly tolerable[2,4] if the granularity is 64B (standard cache-line size). Therefore, we split the page of 4KB to 64B blocks.

This bitmap can be located in the memory separately from the page table. The TLB inside the processor is extended to also hold the address of this bitmap, associated with every page. On the first transactional read or write to the page, if the address of the TB for the page is uninitialized (equal to zero), the processor interrupts the OS and signals that it needs the TB for the page. The OS allocates the space and loads the new TLB entry into the processor, that is now also holding a pointer to the TB. From now on, the processor reads and updates this TB on transactional access to the page blocks.

The total occupied space by both TST and TB grows linearly with the number of processors in the system, and this dependency can be seen graphically in the Fig.2.

As an example, let us assume non-conflicting transactional read and write to a block by e.g. processor 3. After reading from this block, the 3rd bit of TXID for this block will be 1. After writing to the block, the M bit will be 1 and 3rd bit of TXID for this block will be 1. With this simple approach it is easy to quickly detect the conflicts later.

The M and TXID bits are consulted on every (transactional or not) read or write to the block.

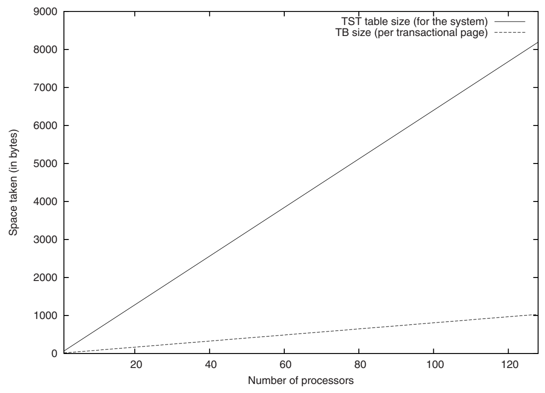


Fig. 2. Space taken for the transactional bookkeeping as a function of the number of processors in the system

TB is actually associated with the physical page in the system, but is accessed only from the virtual addresses, during the translation from the virtual to the physical address. Therefore, we don't need the same number of the TBs as the number of transactionally accessed physical pages in the system. Additionally, many virtual pages can point to the same Transactional Bitmap. This allows inter-process shared memory communication, where many processes share the same TB for different virtual address spaces.

2.2 Hardware

Translation Lookaside Buffer. Each entry in the TLB inside the processor is extended with one **additional bit, T**, that is actually appended to the virtual address on every TLB lookup (Fig. 3). This bit signifies if the TLB entry holds

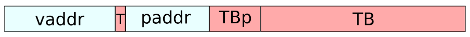


Fig. 3. New TLB entry

the primary (T=0) or the secondary (T=1) copy of the page. The processor, itself, knows whether the lookup it wants to make is the transactional one or not. One more addition to each entry are the **pointer to the Transactional Bitmap, TBp**, and the value of **TB**, whose functionalities are explained in 2.1.

A specialized hardware that can be used to process the M and TXID bits, and for conflict detection, can be seen in the Fig.4. The multiplexer for the M bits selects the proper bit with the value of **bl**, the cacheline block offset inside the page. The TXID is the array of bits, with 64 chunks of 'number of processors' bits. From each of these chunks, for conflict detection each processor is using only one bit, which is hardcoded to the multiplexer input. The multiplexers with 64 input bits and one output bit, can be multilevel to reduce the fan-in.

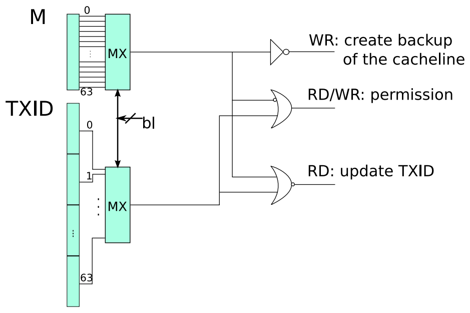


Fig. 4. Additional hardware per TLB entry, for reading and processing of M/TXID bits

Effect on Non-transactional Reads and Writes. The OS flushes the TLB entry from all processors that might hold it, in the case when a pointer to TB (TBp) is changed. This ensures strong atomicity. On a conflict between non-transactional and transactional code, processor running non-transactional code determines that the transaction created a conflict, and either sends a kill_tx (abort your transaction) message to a remote processor if conflicting transaction is INSIDE_PROC or raises an ABORT interrupt if the transaction is INACTIVE.

3 Transactional Access

3.1 Begin Transaction

The transaction begins with the call to the ISA instruction (see Table2) **btx TSTp**, where TSTp is the address of the first entry in the Transaction State Table. Each processor has a unique number, CPUID or TXID, that in our implementation goes from 0 to 31. The processor locates its entry in TST, and sets the value of the *active* field for this processor as INSIDE_PROC, and the *status* field of the transaction as RUNNING. Then it increments the *nesting depth*. After this the processor effectively enters the transactional mode and all memory reads and writes are implicitly transactional.

When the processor changes priority level (e.g. from user mode changes to kernel mode or vice-versa), or when it enters a fault: interrupt, page fault, etc. the *active* field is automatically set to INACTIVE. This is done by writing to the memory bit. Most of the time this will be just a private cache write, unless this TST entry had to be evicted.

On subsequent calls to the **btx**, the processor simply increments the *nesting depth* of the transaction.

3.2 Transactional Read

On every transactional read, the processor consults the TLB entry for the page for the permission to read. If the TBp is zero (i.e. there is no TB associated

Table 2. ISA extensions

begin transaction	btx TSTp
commit transaction	ctx TSTp
clean TB for the TXID	clean_tb vaddr, txid
unblock TXIDs blocked on the given TXID and set the TST free for the given TXID	finish_tx txid
undo the writes and clean the TB for the given TXID	undo_tx vaddr, txid

with this virtual address), the processor raises an interrupt to the OS to create it. OS allocates the space for the TB, then loads the TBp and the TB into the TLB entry and returns to the same instruction that raised the interrupt. A non-transactional read would not raise an interrupt when the TBp is zero in the TLB entry. This is the only difference between the two of them. If there is non-zero TBp, it is obeyed in both cases.

For avoiding the potential race condition when multiple processors wants to read/write to the same block, testing and setting of M and TXID bits should be done atomically.

If **M[bl]=1** for the cacheline, some processor transactionally wrote to it. The read is either permitted or denied, depending on the TXID bits for the cacheline. If some other processors transactionally wrote to this address (bit for this processor is not set), the conflict resolution protocol takes place and read is denied.

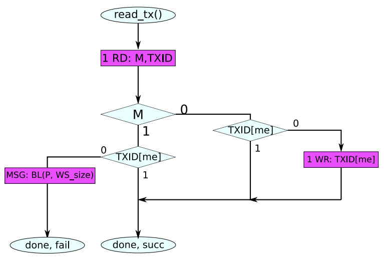
If **M[bl]=0**, read from the cacheline is allowed. If needed (determined by the TLB hardware), the TB entry for the cacheline is updated.

When the processor (e.g. P0) detects a conflict with other processor (e.g. P1), it first locates the P1's entry in TST, then reads the *active* bit of TX1. If TX1 is INACTIVE (e.g. P1 is inside the trap), then P0 sets the *status* of P1 to ABORTED and calls the OS function to initiate the abort of P1's writes. If TX1 has *active* flag set to INSIDE_PROC, then P0 sends an inter-processor interrupt (IPI) to P1 **block_tx(my_writeset_size)**, and waits. Upon receiving the interrupt, the processor P1 compares the size of the write-set of the other transaction with the size of the write-set of himself, and based on that decides which one is going to proceed. For more details about the protocol see 3.6.

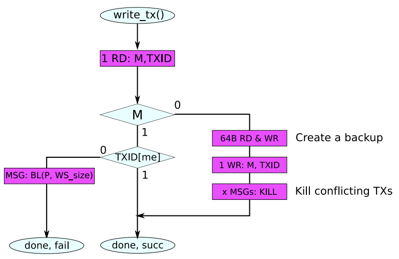
Overhead of each transactional read:
The cost of the transactional read is not uniform as can be seen in Fig. 5a. The first read from the transaction is slowed down by one additional write to TB, and every transactional read after that is slowed down by only one additional read to the Transactional Bitmap, which should be located in cache of the processor.

3.3 Transactional Write

On every transactional write, the processor consults the TLB entry for the permission to write. For avoiding the potential race condition when multiple processors wants to read/write to the same block, testing and setting of M and TXID bits needs to be done atomically.



(a) The overhead of the transactional read



(b) The overhead of the transactional write

Fig. 5. The overhead of the transactional access

If the block was *not* modified (**M bit is zero**), then the processor has to copy the current value of the cache line to the secondary page and set the M and its TXID bits for this block. It also needs to inform other processors, that have read from this block, that they need to restart their running transactions, by sending the **kill_tx** (abort your transaction) to each of them or raising the interrupt ABORT TXn (see 3.5) in case they were *INACTIVE.* Then it writes to the destination cache line and increments the private *writeset_size* variable, used for the conflict resolution.

If the **M bit is one** and the *TXID is not equal to the ID of this processor*, a **conflict resolution** protocol needs to applied. The processor sends the other processor an interrupt **block_tx(my_writeset_size)**, and waits. The other processor, on this interrupt, enters the *block_tx* procedure. For further details about the conflict resolution mechanism see 3.6. On re-writing by the same processor, there is no extra overhead.

Obviously, there is a need for the translation of one virtual address to two or more physical addresses. This is accomplished with the different Page Directory Base Registers (PDBR). Standard PDBR in x86 architectures is the CR3 register. Alternatively, transactional PDBR is defined uniquely for every transaction, in the TST (see 2.1).

The cost of transactional write is not uniform as can be seen in Fig. 5b. First write to the block is slowed down by a write to TB, creating a backup of the block in the secondary page and notifying all dependent transactions/processors to abort transactions. Every next write by the same transaction to the same block is slowed down by only one extra read.

3.4 Commit Transaction

A call to **commit_tx()** is translated to the processor instruction ***ctx TST*p**, which decrements a transaction nesting depth. If the nesting depth is not yet zero, there is no side-effect of instruction call. If the nesting depth becomes zero, then this is the outermost commit (when transactions are nesting). In that case, the processor executes the OS function "**commit(TXn)**". In this function, the OS iterates through the list of all secondary pages and for each of them calls the instruction **clean_tb vaddr, TXID**, in which the processor cleans all the bits for the given TXID. When the OS finishes iterating, it executes *finish_tx*, which informs every transaction blocked on this one that they can proceed, and sets the transaction state in the TST as FREE and INACTIVE. The cleaning of the transactional VPT can be done by other idle processor.

3.5 Abort Transaction

Abort transaction is implemented as an interrupt, to allow non-transactional reads and writes that have a conflict with a transactional write, to undo the transaction, restore original value of the block and then proceed with execution. In the global TST it sets the transaction status to ABORTED, loads the correct virtual page table base address into the PDBR (CR3 register on the x86 architecture) of the chosen processor and then starts iterating through the Transactional VPT, and for every page in it issues a processor the instruction ***undo_tx vaddr, cpuid***, which for each block (bl) in the page: restores a backup copy if it exists, and clears the M/TXID bits of the block (same activity as for clean_tb)

When the OS finishes iterating through the Transactional VPT, it executes *finish_tx*, which informs every transaction blocked on this one that they can proceed, and sets the transaction state in the TST as FREE and INACTIVE.

3.6 Conflict Resolution: Block Transaction IPI

Conflict resolutions are usually done either by the system clock, by virtual time (timestamp), by transaction size, by transaction execution time, or other criteria. We have adopted the conflict resolution by write-set size. Whenever a transaction modifies a block for the first time (and creates a backup value of the block), it increments the private writeset_size counter. The value of this counter is used to determine the priority of transactions. The transaction with bigger value of writeset_size has the higher priority in a conflict.

When a processor, for instance P1, receives the **block_tx (writeset_size)** IPI from the other processor, for instance P0, it compares the write-set size of P0 (P0_writeset_size) with the write-set size of P1 (P1_writeset_size), and if P0_writeset_size is greater then the P1_writeset_size, it aborts (retries) the current transaction. Otherwise, if P0_writeset_size is less or equal to the P1_writeset _size, then P1 puts the P0 into the waiting list. In that case, P1's write-set is greater in size, and the other processor should wait until the completion of transaction inside P1.

This protocol gives the priority to the transactions with the bigger write set. At the same time, it orders transactions so that dead-lock is avoided.

4 Conclusions and Future Work

We presented a new synergistic hardware-software solution for providing Unbounded Page based Hardware Transactional Memory. Leveraging existing Virtual Memory mechanisms, it should allow constant transaction execution speed, regardless of the transaction size, and with small overhead for transactional reads and writes. It does not require any changes of the currently highly-optimized caches. It requires some relatively small changes in current Operating System implementations, to allocate and manipulate the memory space for bookkeeping and secondary pages.

References

1. Bobba, J., Moore, K.E., Volos, H., Yen, L., Hill, M.D., Swift, M.M., Wood, D.A.: Performance pathologies in hardware transactional memory. In: ISCA, pp. 81–91 (2007)
2. Chuang, W., Narayanasamy, S., Venkatesh, G., Sampson, J., Biesbrouck, M.V., Pokam, G., Calder, B., Colavin, O.: Unbounded page-based transactional memory. SIGARCH Comput. Archit. News 34(5), 347–358 (2006)
3. Hammond, L., Carlstrom, B.D., Wong, V., Hertzberg, B., Chen, M., Kozyrakis, C., Olukotun, K.: Programming with transactional coherence and consistency (tcc). In: ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems, pp. 1–13. ACM Press, New York (2004)
4. McDonald, A., Chung, J., Chafi, H., Cao Minh, C., Carlstrom, B.D., Hammond, L., Kozyrakis, C., Olukotun, K.: Characterization of tcc on chip-multiprocessors. In: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (September 2005)
5. Moore, K.E., Bobba, J., Moravan, M.J., Hill, M.D., Wood, D.A.: Logtm: Log-based transactional memory. In: Proceedings of the 12th International Symposium on High-Performance Computer Architecture, pp. 254–265 (February 2006)

Auto-parallelisation of Sieve C++ Programs

Alastair Donaldson[1], Colin Riley[1], Anton Lokhmotov[2,⋆], and Andrew Cook[1]

[1] Codeplay Software
45 York Place, Edinburgh, EH1 3HP, UK
[2] Computer Laboratory, University of Cambridge
15 JJ Thomson Avenue, Cambridge, CB3 0FD, UK

Abstract. We describe an approach to automatic parallelisation of programs written in Sieve C++ (Codeplay's C++ extension), using the Sieve compiler and runtime system. In Sieve C++, the programmer encloses a performance-critical region of code in a *sieve* block, thereby instructing the compiler to delay side-effects until the end of the block. The Sieve system partitions code inside a sieve block into independent fragments and speculatively distributes them among multiple cores. We present implementation details and experimental results for the Sieve system on the Cell BE processor.

1 Introduction

Computer systems are increasingly parallel and heterogeneous, while programs are still largely written in sequential languages assuming a single processor connected to uniform memory. The obvious suggestion that the compiler should automatically distribute a sequential program across the system usually fails in practice because of the complexity of dependence analysis in the presence of *aliasing*.

In Codeplay's Sieve C++ [1,2,3], the programmer can place a code fragment inside a special *sieve* block, thereby instructing the compiler to *delay* writes to memory locations defined outside of the block (global memory) and apply them *in order* on exit from the block. For example, by writing:

```
float *pa, *pb; ...
sieve { // sieve block
  for(int i = 0; i < n; ++i) {
    pb[i] = pa[i] + 42;
  }
} // writes to pb[0:n-1] happen on exit from the block
```

the programmer requests to delay the writes to global memory locations referenced by `pb[0]`,..., `pb[n-1]` until the end of the block. In this example, we can also say that the programmer requests the semantics of the Fortran 90 vector notation

```
pb[0:n-1] = pa[0:n-1] + 42;
```

in which all the reads happen before all the writes [4]. (The vector notation semantics departs from the conventional one if vectors `pa[0:n-1]` and `pb[0:n-1]` overlap.)

⋆ This author gratefully acknowledges the financial support provided by a TNK-BP Cambridge Kapitza Scholarship and by an Overseas Research Students Award.

It is easy to see that the sieve semantics is equivalent to the conventional semantics if code within a sieve block does *not* write to and then subsequently read from a global memory location (otherwise, delaying the write violates the true dependence). The compiler preserves the order of writes to global memory by recording the writes in a FIFO queue and applying the queue on exit from the sieve block.

The sieve construct is different from Software Transactional Memory: code within an atomic block can immediately read new values of modified free variables; code within a sieve block "commits" its side-effects without retrying [2].

Using sieve blocks is attractive for several reasons. First, in order to parallelise code in a sieve block, the compiler needs to conduct dependence analysis only on memory locations defined within the block (local memory). Second, global memory can be read on entry to the sieve block and written to on exit from the block. This maps well to a natural programming style for heterogeneous systems with hierarchical memory. Third, the compiler can distribute the computation across the system speculatively (for example, if the number of iterations is not known at compile time). Results from excessive computation can simply be discarded when committing the side-effect queue to global memory. Fourth, the sieve semantics is deterministic, hence program behaviour is predictable and repeatable.

In this paper, we describe the Sieve C++ iterator and accumulator classes (§2), speculation in the Sieve system (§3), and present implementation details and experimental results for the IBM/Sony/Toshiba Cell BE processor (§4).

2 Syntax

2.1 Sieve and Immediate Functions

A function called from inside a sieve block must be explicitly specified as either *sieve* or *immediate*. Sieve functions can *only* be called from inside sieve blocks or other sieve functions (sieve *scopes*), and have their writes to global memory delayed. Immediate functions can be called from both sieve and non-sieve scopes, and must *not* update global memory. The compiler enforces the correct usage of these function types.

2.2 Iterator Classes

In C/C++, the induction variable and increment value for a loop can be changed within the loop body [4]. Sieve C++ defines special *iterator* classes to track changes to induction variables in order to facilitate speculation.

A simple iterator class has a private *state* variable and a method for updating this variable with the value it should have after a given number of loop iterations.[1] For convenience and efficiency, the class can also include a method for updating the state with the value it should have at the next iteration. All methods updating the state must be specified with the **`update`** keyword. All other methods must be specified as **`const`**, and other fields as private and immutable.

[1] The programmer is responsible for the correct behaviour of this method, as this is not checked by the compiler.

For example, an integer counter class can be defined as:

```
iteratorclass intitr {
  int cnt; // state variable
public:
  immediate intitr(const int cnt) { this->cnt = cnt; }
  immediate operator int() const { return cnt; }
  // update methods
  update void operator +=(const int x) { cnt += x; }
  update void operator ++() { ++cnt; }
}
```

and used in the vector addition example as follows:

```
for(intitr i(0); i < n; ++i) {
  pb[i] = pa[i] + 42;
}
```

Iterator classes are not confined to basic induction. Consider another example:

```
double opt[n]; const double up = 1.1;
for (int i = 0, double Si = 1000.0; i < n; ++i) {
  opt[i] = Si;
  Si *= up;
}
```

Here, the value of variable `Si` (in all iterations but the first) depends on its value at the previous iteration. The programmer can re-write this loop in Sieve C++:

```
sieve {
  powitr Si(1000.0, 1.1);
  for (intitr i(0); i < n; ++i) {
    opt[i] = Si;
    Si.mulUp();
  }
}
```

where `powitr` is defined as follows:

```
iteratorclass powitr {
  double val; // state variable
  const double up;
public:
  immediate powitr(const double val, const double up)
                  { this->val = val; this->up = up; }
  immediate operator double() const { return val; }
  // update methods
  update void mulUp(const int x) { val *= pow(up, x); }
  update void mulUp() { val *= up; }
}
```

The parameterised `mulUp` method can be used to update the state variable `val` with the value it should have after `x` iterations.

2.3 Accumulator Classes

Reduction is the process of obtaining a single element by combining the elements of a vector [4]. The following example computes the sum of the elements of `pa`:

```
float sum = 0.0;
for(int i = 0; i < n; ++i) {
  sum += pa[i];
}
```

If we assume that addition is associative, this reduction can be performed as a number of partial sums the results of which are summed to give the final result.

This computational pattern is supported in Sieve C++ with special *accumulator* classes, which have a distinguished parameterised method called the *merge rule* that is specified with the > symbol (*c.f.* using ~ for destructors in C++).

The floating point accumulator class:

```
accumulatorclass floatsum {
  float acc;
public:
  immediate floatsum() { this->acc = 0.0; }
  >floatsum(float * res, const floatsum **  resv,
                         const unsigned int resc) {
    *res = resv[0]->acc;
    for(int i = 1; i < resc; ++i) { *res += resv[i]->acc; }
  }
  immediate void operator += (float x) { acc += x; }
}
```

can be used to re-write the summation reduction in Sieve C++:

```
sieve {
  floatsum fsum() merges sum;
  for(intitr i(0); i < n; ++i) {
    fsum += p[a];
  }
} // the merge rule '>' is implicitly called here
```

Each partial sum is accumulated into a (private to each core) `acc` variable via the `+=` operator. On exit from the block, the sieve runtime calls the merge rule `>floatsum` to obtain the final result.

Accumulators can be defined for any associative operation. Practical examples include the sum, product, exclusive-or, min and max operators.

3 Speculative Execution

3.1 Split Points

The Sieve system uses the notion of a *split point* to parallelise C++ code within a sieve block. Split points are program points which delimit a sieve block into fragments which can be executed independently, and thus deployed across multiple cores. Split points can either be inserted implicitly by the compiler or explicitly by the programmer via the **`splithere`** keyword. Every sieve block has two implicit split points: at the start and end of the block.

Annotating the vector addition example with a split point inside the loop:

```
sieve { // implicit split point (1)
  for(intitr i(0); i < n; ++i) {
    splithere; // explicit split point (2)
    pb[i] = pa[i] + 42;
  }
} // implicit split point (3)
```

indicates to the compiler that it would be sensible to parallelise this loop.

We can view split points *statically* or *dynamically*: there are three static split points in the above example, as indicated in the comments. Dynamically there are $n + 2$ split points: at the start and end of the sieve block, plus a split point for each iteration of the loop. We can distinguish each dynamic split point in a loop nest by combining the associated static split point with an *iteration vector* (*IV*) [4], comprised of the values of iterators controlling the loop nest. Execution of a sieve block can then be organised as a chain of dynamic split points. For the above example we have the chain $1\langle\ \rangle$, $2\langle 0\rangle$, $2\langle 1\rangle$, . . ., $2\langle n-1\rangle$, $3\langle\ \rangle$ (where $\langle\ \rangle$ denotes an empty *IV*).

A *fragment* is any contiguous portion of a chain of dynamic split points. A sieve block can be efficiently executed by dividing its associated chain of dynamic split points into fragments and executing these fragments in parallel. Each fragment maintains a queue of side-effects which are applied to global memory in order on exit from the block. In addition, each fragment maintains a local accumulator variable for every accumulator declared within the sieve block. On exit from the block the values of these accumulators are merged into appropriate global data structures via the accumulator merge rules.

Note that a fragment typically spans multiple split points. The compiler and runtime system decide how large each fragment should be for the given code and parallel hardware.

3.2 Speculative Execution

A fragment can be described by specifying a static split point, an *IV* giving the values of iterator variables at the start of the fragment, and an integer specifying how many split points should be traversed during the fragment. The result of executing a fragment can be described by: a queue of side-effects to global memory, a set of values for accumulator variables, and an *IV* giving the values of iterator variables at the end of the fragment.

Parallel execution of a loop in a sieve block can be achieved by *guessing* the *IV*s for a sequence of contiguous fragments. Since the first fragment always begins at the start of the sieve block, its *IV* is empty and thus trivial to guess. The runtime system assigns this fragment to one core for execution. The runtime then uses a strategy to *guess* the value which the *IV* will have at the end of this fragment. This guessed vector is used to generate a fragment for parallel execution by another core. If more cores are available, then this guessing process is extended so that each core executes a fragment starting with a guessed *IV*.

If the guessed *IV* for a fragment matches the actual final *IV* of the previous fragment, then the fragments are contiguous, and the guess is correct, or *valid*. Given a chain of correctly guessed fragments, applying the side-effects for each fragment to global memory in sequence has the same effect as executing the fragments in serial with the sieve semantics. If the *IV* for a fragment is incorrectly guessed then its side-effect queue (and the queues of its subsequent fragments) must be discarded. We refer to the execution of fragments with guessed *IV*s as *speculative execution*, since the execution results may have to be discarded if guessing turns out to be wrong.

3.3 Examples

We illustrate the idea of *IV* guessing and speculative execution using the vector addition example. Suppose that at runtime `n` = 20, so that there are 22 dynamic split points: $1\langle\ \rangle$, $2\langle 0\rangle$, $2\langle 1\rangle$,. . ., $2\langle 19\rangle$, $3\langle\ \rangle$. Suppose further that the arrays `pa` and `pb` both have size 20 and that before execution of the loop `pa` is set up so that `pa[`i`]=`i, for any $0 \leq i < 20$.

The following table shows a *perfect* guessing chain for execution of the loop on a quad-core machine:

core	guessed *IV*	guessed length	actual length	side-effects	final *IV*
1	$1\langle\ \rangle$	6	6	`pb[0:4]=[42..46]`	$2\langle 5\rangle$
2	$2\langle 5\rangle$	5	5	`pb[5:9]=[47..51]`	$2\langle 10\rangle$
3	$2\langle 10\rangle$	5	5	`pb[10:14]=[52..56]`	$2\langle 15\rangle$
4	$2\langle 15\rangle$	5	5	`pb[15:19]=[57..61]`	$3\langle\ \rangle$

The guessing chain is perfect because, for $i > 1$, the guessed *IV* for core i matches the final *IV* for core $i - 1$; the guessed length of each fragment matches the actual length of the fragment execution; computation is balanced as evenly as possible between the cores, and no unnecessary computation is performed.

On the other hand, the following table illustrates a poor guessing chain for the same execution:

core	guessed *IV*	guessed length	actual length	side-effects	final *IV*
1	$1\langle\ \rangle$	12	12	`pb[0:10]=[42..52]`	$2\langle 11\rangle$
2	$2\langle 11\rangle$	12	9	`pb[11:19]=[53..61]`	$3\langle\ \rangle$
3	$2\langle 23\rangle$	12	1	`pb[23]=`$\perp$	$3\langle\ \rangle$
4	$2\langle 35\rangle$	12	1	`pb[35]=`$\perp$	$3\langle\ \rangle$

In this example core 1 performs most of the computation, and a correct *IV* guess allows core 2 to do the rest of the computation in parallel. The fact that the guessed fragment length for core 2 is too large does not affect correctness of execution: when the loop condition becomes false, this core reaches the end of the sieve block as expected. However, the guessed *IV*s for cores 3 and 4 are based on the expected fragment length for core 2. As a result, these cores attempt to read subscripts of `pa` and write to subscripts of `pb` which are beyond the bounds of these arrays. The resulting side-effects are marked grey in the above table, and the undefined values speculatively assigned to `pb[23]` and `pb[35]` are denoted $\perp$.

After this speculative execution the runtime assesses the correctness of its guessing effort. It determines that cores 1 and 2 have performed the required loop execution, and applies their side effects to global memory (filling the array `pb`). The runtime also detects that the guesses for cores 3 and 4 were incorrect and therefore does *not* apply their side-effect queues. As a result, although this poor guessing effort does not lead to optimal exploitation of parallel hardware, it still results in correct, deterministic execution of the sieve block.

3.4 Coping with Invalid Guesses

In the above example, attempting to read from `pb[23]` or `pb[35]` may result in an access exception. In more complicated examples, speculative execution could result in other exceptions (e.g. division by zero). These exceptions are caught by the runtime system and hidden until the runtime can determine whether the guess which caused a given exception is valid (i.e. whether this guess simulates serial behaviour). If a guess turns out to be invalid (as in the above example) then, in addition to discarding the side-effect queue, any exceptions arising from the speculated execution are also ignored. If the guess is valid, the runtime system will re-run the fragment and this time expose the exception so that the user can debug the program as usual.

3.5 Advanced Techniques for Guessing

As discussed in §2.2, the **`update`** methods provide a way to set up the state of iterator variables as if a given number of loop iterations had already been executed. Nevertheless, sometimes it is impossible to determine the number of iterations a given loop will execute: the loop may exit early via **`break`** statements; the loop bounds may change dynamically, *etc.* Thus, good guessing is a challenge.

A simple guesser can operate by running small fragments (e.g. with a length of one) to check for updates to iterator variables in the loop body. Once the pattern of these updates is discovered, the runtime can make larger guesses to ensure that the computation within each fragment is sufficient to outweigh the runtime overhead of managing the fragments.

More advanced speculation techniques could be employed by having the compiler communicate extra sieve block meta-data to the runtime. For example, the compiler could identify the split points which a given iterator spans, and mark split points across which no iterators are live, allowing speculation before and after these points to be independent. This would ease the task of checking guess-chain correctness, and increase the likelihood of valid guesses.

4 Implementation on the Cell BE

Experimental results showing the effectiveness of parallelisation via the Sieve system for multi-core x86 systems are presented in [2]. We focus here on implementation and experimental results for the Cell Broadband Engine (BE) processor [5]. Fig. 1 illustrates the Cell BE architecture, which consists of a "power processing element" (PPE) and eight[2] "synergistic processing elements" (SPEs). Each SPE has 256KB local memory, and accesses main memory using DMA transfers.

[2] Our implementation is for the Sony PlayStation 3 console, on which only six of the SPEs are available to the programmer.

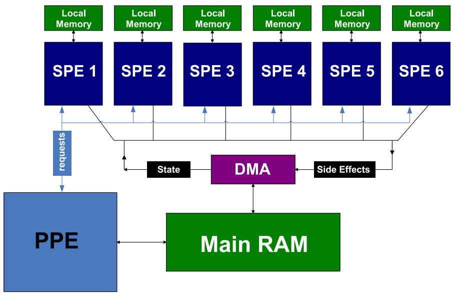


Fig. 1. Architecture overview of the Cell BE processor. (Only the six SPEs which are available to the PlayStation 3 programmer are shown.)

4.1 The Cell Runtime

The Codeplay Sieve system is designed to be easily ported to new processor architectures. The Sieve compiler has an ANSI C backend which can be used to enable support for architectures where a C compiler already exists. In particular, the IBM Cell SDK 2.0 (which we used for our experiments) includes the GCC and xlC compilers.

In addition to C source files, the Sieve compiler also outputs details of which source files should be compiled for which processing elements. This allows auto-generation of a makefile for direct compilation of Sieve compiler's output to a parallelised binary.

A runtime for the Cell BE took only two weeks of time for one developer to create; this runtime incorporates simple loop-level speculation, support of iterator and accumulator classes, an SPE software-managed cache, side-effect queue management, and streaming DMA optimisations.

The runtime manages the SPEs as a pool. The SPEs boot into a tight runtime loop which checks for fragments which are waiting to be executed. The side-effect queue for each SPE is implemented using a ping-pong double-buffered streaming technique to achieve constant memory usage, yet to allow efficient use of SPE-initiated non-blocking DMA operations. Each SPE requires a certain amount of its 256 KB local store to be reserved for runtime use. Reserving too small an amount can lead to communication bottlenecks, as large side-effect queues must be streamed back in smaller chunks.

The PPE produces guesses, which are consumed by the SPEs, and writes SPE side-effect queues to main memory on exit from a sieve block. When an accumulator is used within a sieve block, the PPE is also responsible for collecting accumulated values from the SPEs and merging these values using the accumulator merge rule.

4.2 Experimental Results

Fig. 2 shows the speedup (over a single SPE) for five example Sieve C++ applications: a cyclic redundancy check of a randomly generated 8MB message (CRC); generation of a ray-traced 500×500 image representing a 3D intersection of a 4D Julia set, with reflection and phong shaded lighting (Julia); a noise reduction filter over a 512×512 image, using a 20×20 neighbourhood per pixel (Noise); generation of a 1500×1500 fragment of the Mandelbrot set (Mandelbrot); a 4M-point Fast Fourier Transform (FFT).

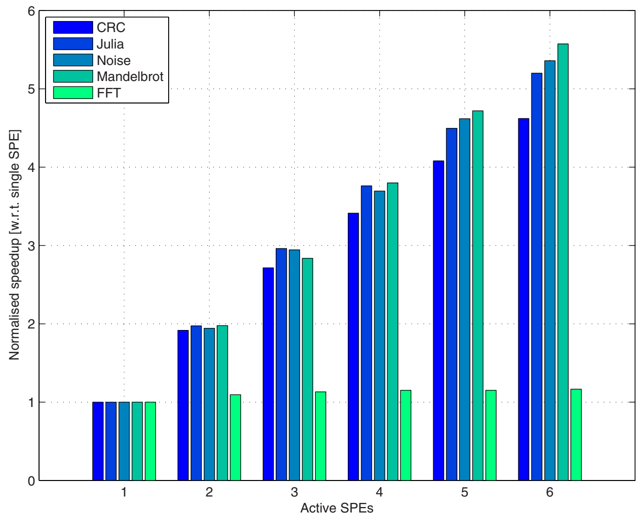


Fig. 2. Scalability results on Sony PlayStation 3

The results show that all Sieve C++ benchmarks, with the exception of FFT, scale well over multiple cores (similar to our results for multi-core x86 systems [2]), with 77%, 86.7%, 89.3% and 92.9% efficiency on 6 SPEs for the CRC, Julia, Noise, and Mandelbrot benchmarks, respectively. Of these scalable benchmarks, the CRC is least efficient because it uses accumulator variables which require a modest amount of serial computation on the PPE to execute the accumulator merge rule. The Julia, Noise and Mandelbrot do not use accumulators, and hence are more efficient. The experiments were performed using the delayed-write combining technique discussed next.

4.3 Combining Writes

Managing the side-effect queue incurs space and time overheads. A side-effect queue element is written as a triple `(address,size,data)`, where `address` is the destination memory address and `size` is the data size (in bytes). In the current implementation, the combination of the address and size is 8 bytes long and data is padded to a minimum of 4 bytes. Thus, a delayed write of a single byte results in writing 12 bytes to the queue.

An easy way to reduce this space overhead is to combine a series of small consecutive writes into a single, larger write. This can be achieved by comparing each write with the last entry in the queue, merging the data and updating the data size information if the data items turn out to be contiguous in memory.

Consider the following sieve block which writes characters to an array:

```
char *p = 0xfeed;
sieve {
  p[0] = 'b'; p[1] = 'e'; p[2] = 'e'; p[3] = 'f';
}
```

Without delayed-write combining, the side-effect queue grows significantly with each delayed write:

```
sieve {
  p[0] = 'b'; // queue: [(0xfeed,1,'b')]
  p[1] = 'e'; // queue: [(0xfeed,1,'b'), (0xfeee,1,'e')]
  p[2] = 'e'; // queue: [(0xfeed,1,'b'), (0xfeee,1,'e'),
              //         (0xfeef,1,'e')]
  p[3] = 'f'; // queue: [(0xfeed,1,'b'), (0xfeee,1,'e'),
              //         (0xfeef,1,'e'), (0xfef0,1,'f')]
}
```

Applying delayed-write combining results in a smaller queue:

```
sieve {
  p[0] = 'b'; // queue: [(0xfeed,1,'b')]
  p[1] = 'e'; // queue: [(0xfeed,2,"be")]
  p[2] = 'e'; // queue: [(0xfeed,3,"bee")]
  p[3] = 'f'; // queue: [(0xfeed,4,"beef")]
}
```

This optimisation is particularly beneficial for the Mandelbrot benchmark which writes pixels into a contiguous **`unsigned char`** array. Computing a 600×600 Mandelbrot image across 6 SPEs (working on 100 rows each) means that each fragment has 60,000 twelve-byte queue entries. Using the optimal transfer size of 16KB implies 44 DMA operations per fragment [5]. Applying delayed-write combining results in only 4 DMA operations per fragment (three 16KB transfers followed by a transfer of 10,872 bytes). The total number of DMA operations is thus reduced from 264 to 24. The benefit is in having less of a bus-bottleneck and less blocking whilst ping-ponging buffers (as the queues now take longer to fill). In our experiments, using delayed-write combining resulted in 21.4% faster execution time when computing a 1500×1500 pixel image.

5 Conclusion

We have presented the Sieve compiler and runtime system for auto-parallelising Sieve C++ programs. Our future work will focus on advanced implementation techniques of Sieve C++ programs for performance and scalability on the Cell BE and other multi-core architectures.

References

1. Codeplay: Portable high-performance compilers, `http://www.codeplay.com/`
2. Lokhmotov, A., Mycroft, A., Richards, A.: Delayed side-effects ease multi-core programming. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 629–638. Springer, Heidelberg (2007)
3. Lindley, S.: Implementing deterministic declarative concurrency using sieves. In: Proc. of the ACM SIGPLAN Workshop on Declarative Aspects of Multicore Programming (DAMP), ACM Press, New York (2007)
4. Allen, R., Kennedy, K.: Optimizing Compilers for Modern Architectures. Morgan Kaufmann, San Francisco (2002)
5. IBM/Sony/Toshiba: Cell Broadband Engine Programming Handbook Version 1.1 (2007)

Adaptive L2 Cache for Chip Multiprocessors

Domingo Benítez[1], Juan C. Moure[2], Dolores I. Rexachs[2], and Emilio Luque[2]

[1] DIS Department and IUSIANI, University of Las Palmas G.C., Spain
dbenitez@dis.ulpgc.es
[2] CAOS Department, University Autónoma of Barcelona, Spain
{JuanCarlos.Moure,Dolores.Rexachs,Emilio.Luque}@uab.es

Abstract. An open question in chip multiprocessors is how to organize large on-chip cache resources. Its answer must consider hit/miss latencies, energy consumption, and power dissipation. To handle this diversity of metrics, we propose the Amorphous Cache, an adaptive heterogeneous architecture for large cache memories that provides new ways of configurability. The Amorphous Cache adapts to fit the code and data by using partial array shutdowns during run-time. Its cache configuration can be resized and the set associativity changed. Four reconfiguration modes can be used, which prioritize either IPC, processor power dissipation, energy consumption of processor and DIMM memory module, or processor power$^2 \times$delay product. They have been evaluated in CMPs that use private L2 caches and execute independent tasks. When one of the cores of a CMP with 4-MB L2 shared-cache is used as baseline, the maximum average improvements in IPC, power dissipation, energy consumption, and power$^2 \times$delay achieved by a single core with 2-MB private L2 Amorphous Cache are 14.2%, 44.3%, 18.1%, and 29.4% respectively.

1 Introduction

Future chip multiprocessors (CMPs) will require innovative designs of on-chip memory hierarchies. One of the options in the design of a CMP for taking advantage of the increase of transistor count in the chip consists in including a larger cache [5]. Increasing on-chip wire delays and leakage power will influence on how this large storage should be organized if latency and power must be minimized.

Based on the observation that the aim of a large cache is to reduce memory related stalls rather than to reduce misses, and that the best cache configuration depends on program and design constraints, we propose a reconfigurable architecture for large on-chip cache memories called Amorphous Cache. Our contributions are the following: [A] We introduce a novel reconfigurable cache that uses partial array shutdowns during program execution (Section 2). It adapts the best cache architecture to the needs of different programs after chip fabrication. [B] Several actual layouts were created using a 130-nm CMOS technology in order to determine the impact of reconfiguration logic on architectural metrics (Section 3). [C] Four reconfiguration modes are defined, which prioritize one of a set of architectural metrics: IPC, processor power dissipation, energy consumption of processor and DIMM memory module, or processor power$^2 \times$delay product. These modes offer distinct trade-offs between performance improvement and energy usage. They have been evaluated in CMP processors that use private L2-caches and execute independent tasks. When one of the cores of a CMP with 4-MB L2 shared-cache is used as baseline, a core with 2-MB private L2 Amorphous Cache can provide at the same time higher performance, lower power dissipation and lower energy consumption (Section 5). Other three sections describe the experimental methodology, related work and conclusions.

2 Reconfigurable Cache

In this section, we describe a specialized reconfigurable circuit for CMP caches (shared or private) called *Amorphous Cache (AC)*. The term amorphous is motivated by the fact that the range of cache configurations, critical paths and power consumptions is not homogeneous. The integrated circuit is organized into an array of heterogeneous blocks called *subcaches*, which are selectively connected to the external ports by programmable pass gates called *interconnection gates*. All disconnected sub-caches are powered-off. This circuit includes configuration registers that store *configuration bits*. The AC cache implements different cache organizations by overwriting the configuration registers. Fig. 1 shows the basic architecture of a 2-MB Amorphous Cache.

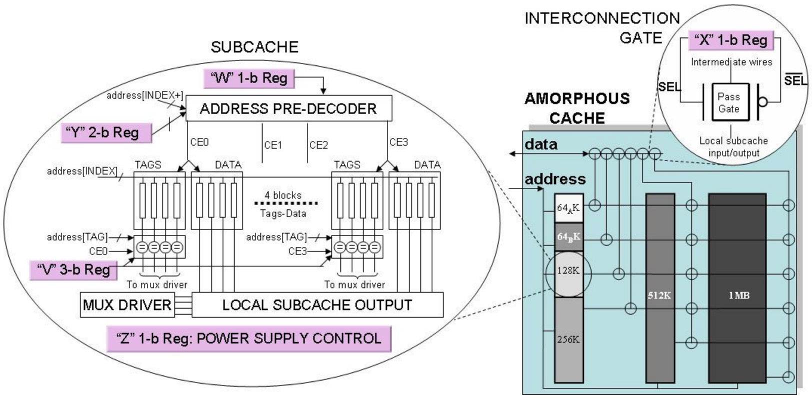


Fig. 1. Structure of the reconfigurable Amorphous Cache

An interconnection gate is composed of a pass gate and the *X 1-bit register*. It connects or disconnects the local port of a subcache to intermediate wires, which are connected to the input/output port of the AC cache. We propose using the programmable interconnection gates to selectively aggregate the heterogeneous sizes of subcaches to adapt the cache to the needs of program working-sets.

The subcaches are based on the classical organization of CMOS cache memories. Each of them consists of four cache subarrays called *tags-data*, which are 4-way set-associative caches with 256-byte blocks. The *Y 2-bit register* determines its set associativity. Three different set associativities can be selected:

4-, 8- and 16-way, while the full capacity of the subcache is utilized. Depending on the address bits *address[INDEX+]*, one tags-data subarray is activated by one of the address pre-decoder outputs (*CE*) and acceded with the common index: *address[INDEX]*. We assume that the critical paths of the tags-data subarrays are the same as proposed by Zhan et al [16]. They argued that by increasing the sizes of certain transistors, the critical path inside the tag and data arrays is not affected by the pre-decoder output signals. The address bits connected to the pre-decoder and tag-data subarrays depend on each subcache because different subcaches offer different sizes. The cache tag bits *address[TAG]* used to check for a match also depend on the cache configuration. The *V 3-bit register* indicates which tag bits are used to determine hit or miss.

Our design takes into account six subcaches with sizes that vary from 64 KB to 1 MB. When all subcaches are powered on, which is indicated by the *W 1-bit register*, the total cache size is 2 MB. In this paper, we evaluate 18 tuneable cache configurations of the AC cache with the following range of capacities: from 64 KB to 2 MB, which can be either 4-, 8- or 16-way set-associative. The subcaches that are not required for a determined cache configuration are powered-off by using the *Z 1-bit register*. Each subcache has an independent power ring which can be selectively connected to the voltage supply.

The Amorphous Cache can be reconfigured either dynamically, after each instruction or temporal interval, or statically, before a different program begins fetching instructions. Dynamic reconfiguration is outside the scope of this paper and will be evaluated in future papers. In the rest of paper, we assume that the Amorphous Cache is statically reconfigured. This is done by software in two phases called *Learning* and *Actuation*. The *Learning Stage* is executed for each program during profiling time, in which the software calculates the performance and energy usage from the measures made by internal processor counters, which monitor clock cycles, cache hits, and cache misses. This task is repeated for each tuneable cache configuration. After that, each program is associated with the best cache configuration, which in turn, could be distinct if a different prioritized metric is selected at profiling time: IPC, energy, etc. In the *Actuation Stage*, the prioritized metric is indicated by software, the corresponding cache configuration is picked, and the AC cache is reconfigured before running the program.

3 Physical Implementation

The hit latency, dynamic energy, and leakage power of the tuneable cache configurations vary significantly. We created several VLSI circuits using the standard-cells design methodology to determine as accurately as possible the impact of the extra circuitry of the AC cache on these parameters. Fig. 2 shows the layouts of a 256-KB conventional cache memory and the 256-KB subcache of the AC cache. We used Synopsis Design Compiler [13] to synthesize different VHDL designs into technology-dependent netlists, and Cadence Encounter SoC tools [4] to perform automatic placement and routing of the full-chips from the netlists that allowed us to make the temporal and power analysis of the layouts. The technology we used was UMC 130-nm CMOS with 8-metal layers and 1.2 V power supply [15] through the Europractice university program [6]. A SRAM compiler from Virtual Silicon Technology was used to generate single-port SRAM modules. Using Cadence Encounter, we measured the access time, dynamic access energy, and leakage power of various conventional baseline caches and the equivalent configurations of the Amorphous Cache.

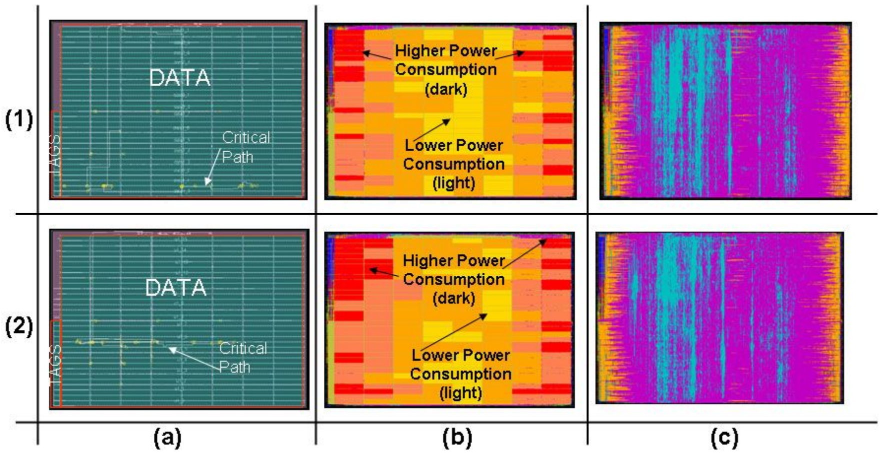

(1)
(2)
(a)
(b)
(c)
DATA
TAGS
Critical Path
Higher Power Consumption (dark)
Lower Power Consumption (light)

Fig. 2. Physical layouts of: (1) a 256-KB 8-way set-associative conventional cache memory; (2) the 256-KB subcache of the Amorphous Cache. (a) Placed SRAM blocks and the critical path of the routed layout, (b) power graph, (c) routed layout.

We observe that the additional transistors required by the Amorphous Cache are mainly used in the tag memories, configuration registers, address pre-decoders, and selection logic at the output of the comparators. The amount of additional transistors in the layout of the 2-MB AC cache relative to the layout of a 2-MB baseline cache is lower than 0.5%. The circuits of the critical paths that are affected by the specialized reconfigurable architecture are the address pre-decoders, comparators, multiplexer drivers, and wire lines connected to the input address and output data.

An architectural model for the AC cache is required for detailed simulations of complete processors. We used the results obtained from the physical layouts to scale the values provided by CACTI 4 [14] for conventional cache memories. Table 1 contains the access time, dynamic access energy and leakage power for each cache configuration of the 70-nm AC cache using a 4-GHz clock. They were obtained by taking the original values provided by CACTI and adding the respective percentages measured in 130-nm.

Table 1. Configurations of a 2-MB Amorphous Cache for a 4-GHz clock and λ=70 nm. Legends: at is cache access time, ae is dynamic access energy, lp is leakage power

AC Configuration	**at [cycles]**	**ae [nJ]**	**lp [mW]**	**AC Configuration**	**at [cycles]**	**ae [nJ]**	**lp [mW]**
64KB 4-way	4	0.12	302.05	512KB 8-way	6	1.48	2555.02
64KB 8-way	4	0.23	313.77	512KB 16-way	7	2.99	2585.60
128KB 4-way	4	0.23	603.77	1MB 4-way	10	1.19	4973.29
128KB 8-way	4	0.46	622.80	1MB 8-way	9	3.00	5033.59
256KB 4-way	5	0.31	1219.07	2MB 4-way	17	3.61	10796.20
256KB 8-way	5	0.60	1259.01	2MB 8-way	17	6.82	11401.20
256KB 16-way	5	1.16	1292.20	2MB 16-way	17	13.38	12085.70

Since the Amorphous Cache is a reconfigurable circuit, it requires temporal and power overheads that are proportional to the number of conguration bits. For the Amorphous Cache shown in Fig. 1, each one of its six subcaches has seven conguration 1-bit registers, in addition to the 21 1-bit registers of the interconnection gates; i.e., 63 1-bit configuration registers determine the cache configuration of the AC. They are loaded before programs begin execution. Assuming that these bits are serially loaded from an on-chip ROM memory with a 800 MHz conguration clock signal, the reconguration time of the Amorphous Cache is 0.08 μs; i.e. 320 penalization cycles for a clock speed of 4 GHz. This temporal overhead was considered in our architectural simulations. Furthermore, we also considered the additional 5 μJ energy consumed by the Amorphous Cache in each cache reconguration. The amount of leakage power dissipated by the configuration registers and associated circuits was assigned the value of 10 mW. In the rest of paper, the Amorphous Cache is evaluated when it is used as private L2 cache memory in a CMP.

4 Experimental Methodology

We evaluated a baseline core in a L2 shared-cache CMP (see Table 2) and a core with private L2 Amorphous Cache. The L1 and L2 caches of the baseline system were carefully selected to reflect the cache configuration of a current dual-core shared-cache CMP. Seven architectural metrics were used: IPC, L2 miss rate, static and dynamic energy consumption, static and dynamic processor power dissipation, and power^2×delay product. We decompose the total energy consumed by a core into the energy consumed by the L2 cache and the energy consumed by the rest of processor. The dynamic energy in each L2 access and the L2 leakage power for each adaptive and non-adaptive cache configuration were provided by CACTI 4 for a 70-nm technology [14] and the AC cache model described above. For the rest of processor, we assume that the energy consumptions are proportional to the respective values consumed by the baseline 4-MB L2 cache. This proportionality for the dynamic energy was determined by using Wattch simulator [3], and factoring its results to reflect the realistic dynamic energy breakdown of the 130-nm 1MB-L2 AMD Athlon 64 processor [11]. Based on the leakage power breakdown of this Athlon processor, we assume that the leakage power dissipated by the rest of processor is *1.5X* times the leakage power dissipated by the baseline L2 cache. Since the baseline processor except L2 remains identical, its dynamic and static energy is considered constant during all experiments. We also use an execution-driven simulator that models the Alpha ISA to determine execution time and cache activity.

Table 2. Baseline out-of-order processor configuration integrated in a CMP

Operating Frequency	4 GHz
CMOS Technology	$\lambda = 70$ nm, Power supply= 1.2 V.
Fetch Engine	Decoupled, pipeline stages: 8, fetch queue: 16-entry
Instruction and Data Cache	16KB+16KB, line-size: 64B, 8-way, LRU, at: 3-cycle, MSHR: 32-entry, 2-port, early-start, ae: 0.037 nJ/access, lp: 70.5 mW
Branch Predictor	Gshare 64K-entry, 2-bit counters; BTB: 512-entry, 8-way; minimum miss-prediction penalty: 17 cycles
Decode/Issue/Retired	Up to 8 instructions per cycle; issue queue: 48-entry; reorder buffer: 256-entry; 8 pipeline stages
Execution Units	Operation latencies like Pentium 4
Load/Store Unit	64/32-entry queue, Perfect memory disambiguation, Store to Load forwarding
Unified and Shared L2 Cache	Inclusive, 4 MB, line-size:256B, 16-way, LRU, at:27 cycles, 1-port, ae:15.38 nJ/access, lp:19570.5 mW
External Bus Unit	at: 20-cycle latency, ae: 5 nJ/external-access
Main Memory	at:380 cycles, ae: 245 nJ/access (DDR2 DIMM, 1.8 V, 333 MHz, 64-bit, pin-out capac.:10 pF/pin), lp:2.2 mW

We used single-threaded SPEC CPU2000, Mediabench II and NAS-2.3 benchmarks compiled for the Alpha ISA with the $-O4$ optimizations. We assume that workloads are multiprogrammed, which are believed to be encountered typically in desktops [5]. This means that each core runs a different single-threaded program. We also assume that no data sharing exists among threads. Table 3 shows the benchmarks and the instruction intervals of the simulation stages. Additionally, Table 3 includes some results that were obtained from the simulations of the baseline processor.

The selection policy of benchmarks was based on two key ideas. On the one hand, they characterize representative workloads because the L2 miss rates (MR) of the baseline processor vary a lot: from 23.1% (ammp) to 1.0E-6 (eon). On the other hand, the IPCs of the baseline processor also exhibit a wide variation: from 3.54 (twolf) to 0.04 (ammp). This allows us to analyze the impact of the Amorphous Cache on performance of a baseline processor that executes the programs with different cache efficiencies.

Table 3. Benchmark summary and performance of the baseline processor. Legends: M/I= Memory accesses per Instruction, IPC= Instructions Per Cycle, MR= number of L2 misses per memory access, B= Billions of instructions, M= Millions of instructions.

Benchmark		**Instruct. Intervals**		**Baseline Processor**			**Benchmark**		**Instruct. Intervals**		**Baseline Processor**		
Name	Suite	Fwd	Sim.	M/I	IPC	MR	Name	Suite	Fwd	Sim.	M/I	IPC	MR
ammp	spec	0	2B	0.25	0.04	23.1%	is	NAS	0	382M	0.35	0.51	0.30%
applu	spec	100M	2B	0.38	0.56	0.85%	lu	NAS	50K	2B	0.39	1.25	0.23%
apsi	spec	0	57M	0.48	3.40	1.4E-5	lucas	spec	0	2B	0.17	1.02	0.89%
art	spec	50K	2B	0.38	0.56	1.0E-5	mcf	spec	1B	2B	0.33	3.19	0.08%
bt	NAS	50K	2B	0.42	0.48	0.97%	mesa	spec	0	2B	0.36	2.60	0.03%
cg	NAS	0	1.6B	0.46	0.23	1.36%	mg	NAS	50 K	2B	0.46	0.96	0.40%
cjpeg	Med2	0	1.6B	0.27	2.57	1.6E-6	mgrid	spec	0	2B	0.37	1.04	0.47%
eon	spec	0	2B	0.46	2.39	1.0E-6	perlbmk	spec	0	2B	0.42	1.96	0.08%
ep	NAS	50 K	2B	0.26	0.97	8.0E-6	sixtrack	spec	0.5B	1B	0.31	2.50	3.8E-5
equake	spec	0	2B	0.33	3.48	4.3E-5	sp	NAS	50K	2B	0.40	0.48	1.01%
ft	NAS	0	1.2B	0.46	0.84	0.28%	swim	spec	0	2B	0.31	0.56	1.15%
galgel	spec	0	2B	0.42	0.72	0.01%	twolf	spec	100M	376M	0.32	3.54	0.01%
gzip	spec	0	2B	0.28	1.42	0.06%	vpr	spec	200M	2B	0.40	1.22	0.18%
gzip	spec	0	2B	0.31	1.21	0.06%	wupwise	spec	0	2B	0.18	2.05	0.19%

5 Application to Adaptive CMPs with Private L2 Cache

We assume that only one single-threaded benchmark is running on one core of the baseline CMP, and all the shared L2 cache is used by the active core. We compare the performance and energy usage of the baseline processor with shared 4-MB L2 cache with a single core of a CMP that uses 2-MB private L2 Amorphous Cache. When the programs running on different cores share neither data nor code, the number of misses in the shared L2 cache of a CMP may be higher than having all the shared L2 cache for only one active core. The hypothesis is that there can be more L2 conflict misses than when all cores except one of them are inactive, due to multiple independent working-sets sharing the same cache. However, private L2 caches are not affected by the activity of single-threaded programs in other private caches. Thus, our results may establish a lower bound for the performance improvement and energy efficiency of CMPs with private L2 AC cache over shared-cache CMPs with double-size L2 cache.

The criterion for determining the best cache configuration of AC cache depends whether the primary objective is optimizing performance or energy usage. This paper evaluates the potential of the private 2-MB L2 AC cache in four different *Reconfiguration Modes*, which prioritize four architectural metrics respectively. The *performance-aware reconfiguration mode* picks the cache configuration that provides the highest IPC for each program. In *power-aware reconfiguration mode*, the simulator uses for each program the cache configuration that provides the lower average power dissipation. We have additionally evaluated selection mechanisms driven by the L2 configuration that provides for each benchmark the lowest energy consumption, called *energy-aware reconfiguration*, or the lowest power^2×delay product, called *time&energy-aware reconfiguration.*

Fig. 3 compares the performance-aware reconfiguration mode of the 2-MB L2 private Amorphous Cache with the 4-MB L2 shared-cache baseline configuration for each benchmark. Benchmarks exhibit IPC increase (positive improvement) or decrease (negative improvement) with respect to the baseline configuration that ranges between +47.9% (ft) and -28.0% (perlbmk). For those benchmarks in which the number of memory stall cycles of the AC cache configuration is larger than baseline (21% of benchmarks: mg, mcf, ammp, lu, sp, perlbmk), a negative performance improvement was always observed. Thus, the reduced L2 hit time of AC cache does not compensate for the product of the increased L2 miss rate and the 400-cycle miss penalty. However, the most frequent case occurs when the L2 hit latency of the AC cache is lower than baseline (79% benchmarks). In these cases, a positive performance improvement was always observed. On average, the performance improvement achieved by our adaptive performance-aware L2 cache is 14.2%, which is due to a combination of the following factors: [A] memory performance limits processor performance when the L2 size is lower than a threshold that is program dependent, [B] the program working-set can use a lower size L2 cache with minimal increase in the L2 miss rate, [C] the out-of-order instruction processing provides a relative low toleration for the memory stall cycles. Therefore, we conclude that for higher performance, it is more efficient to increase the miss rate of the L2 configuration a little (on average 17.7%) by picking a smaller cache size since its impact on performance can be hidden by the reduction in L2 hit latency (on average 63.9%) −see Table 4−.

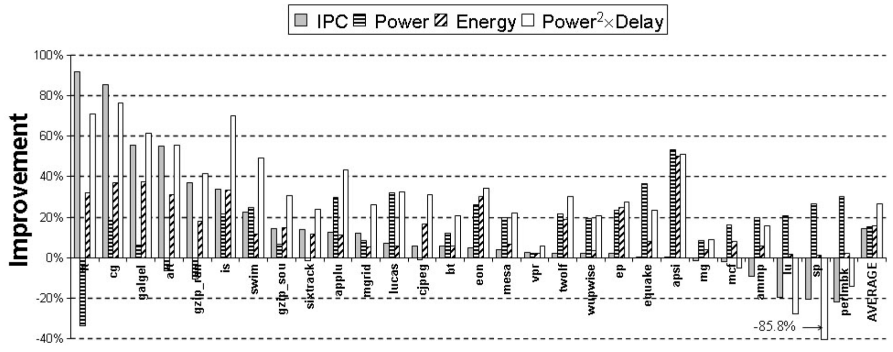


Fig. 3. Results of the performance-aware reconfiguration mode of the 2-MB L2 AC

6 Related Work

Recent studies have considered that wire latency is a primary design factor in CMP caches. Balasubramonian et al proposed a reconfigurable exclusive L2/L3 cache that can be tuned at every instruction interval or subroutine call [1]. They concluded that the L2/L3 cache with configurable scheme can have a profound

Table 4. Average improvements (%) achieved by four reconfiguration modes of the private L2 AC cache relative to the L2 shared-cache baseline system after the simulation of 28 benchmarks. Positive numbers (+) indicate that AC cache achieves improvement in the respective metric. Negative numbers (-) indicate just the opposite.

Reconfiguration Mode	IPC	Energy	Power	$\text{Power}^2 \times$ Delay	L2 Miss Rate	L2 Access Time	Mem Stalls	L2 Dyn Eng	L2 Leak Pow
Performance	+14.2	+15.7	+15.4	+26.5	−17.7	+63.9	+17.7	+77.4	+72.6
Power	−19.1	+6.1	+44.3	−79.4	−143.4	+84.4	−136.6	+99.0	+98.3
Energy	+8.5	+18.1	+25.1	+16.3	−61.0	+79.5	−16.7	+94.8	+91.6
Time&Energy	+12.5	+17.1	+20.8	+29.4	−56.0	+71.6	+12.9	+89.5	+83.3

impact on increasing energy efficiency when compared with a conventional three-level cache hierarchy. Huh et al proposed a low-latency L2 cache for a 16-core CMP, which is organized as a non-uniform cache architecture array (NUCA) with a switched network embedded in it [7]. They conclude that the best L2 cache configuration is a static organization with sharing degrees of two or four. Beckmann et at compared three latency reduction techniques for CMPs with an 8-core shared cache [2]. They observed that combining the three latency reduction techniques can decrease the L2 hit latencies of CMPs. Speight et al studied how CMP L2 caches interact with off-chip L3 caches and how L2 caches temporally absorb modified replacement blocks from other caches [12].

Several researches have investigated dynamically re-allocating cache capacity for CMPs. Liu et al proposed achieving dynamic bank allocation by re-mapping the banks [10]. Iyer proposed priority-based cache management systems to allocate cache resources by OS-assigned priority [8]. To prevent thread starvation to cache capacity sharing, Kim et al investigated fairness issues in CMP cache sharing [9].

7 Conclusions

This paper presents an architecture for heterogeneous adaptive caches called *Amorphous Cache*, which can adapt its size at run-time to match the actual cache requirements of the application working-sets. The evaluation of detailed circuit layouts have provide us realistic limits in the granularity of sub-cache shutdowns, and an estimation of the cost of shutdowns and the fixed penalty due to its reconfiguration capability. Four reconfiguration modes can be used, which achieve different levels of trade-off in the improvements of performance and energy usage. In CMP processors with shared L2 cache, the memory latency relative to cycle time and power consumption both continue to grow, and processors are neither very latency- nor power-tolerant. Using the Amorphous Cache as private L2 cache in a CMP processor has many potential advantages for multiprogrammed and independent workloads; IPC, processor power dissipation, energy consumption of processor and DIMM memory module, and processor power^2×delay product can be improved, on average, by 14.2%, 44.3%, 18.1%, and 29.4% respectively.

Acknowledgements

We sincerely thank Prof. Antonio Nunez and Pedro P. Carballo for their invaluable comments on the designs of the integrated circuits. This work was partially funded by the MEC-Spain under contract TIN2004-03388.

References

1. Balasubramonian, R., Albonesi, D.H., Buyuktosunoglu, A., Dwarkadas, S.: A Dynamically Tunable Memory Hierarchy. IEEE Tran. Comp. 2(10), 1243–1257 (2003)
2. Beckmann, B.M., Wood, D.A.: Managing wire delay in large chip-multiprocessor caches. In: Proc. of the 37th MICRO, pp. 319–330. IEEE Computer Society, Los Alamitos (2004)
3. Brooks, D., Tiwari, V., Martonosi, M.: Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In: Proc. of the 27th Int. Symp. Comp. Arch., pp. 83–94. IEEE Computer Society, Los Alamitos (2000)
4. CADENCE Design Systems, `http://www.cadence.com`
5. Chang, J., Sohi, G.S.: Cooperative Caching for Chip Multiprocessors. In: Proceedings of the 33rd Int. Symp. Comp. Arch., pp. 264–276. IEEE Computer Society, Los Alamitos (2006)
6. EUROPRACTICE, `http://www.te.rl.ac.uk/europractice_com/`
7. Huh, J., Kim, C., Shafi, H., Zhang, L., Burger, D., Keckler, S.W.: A NUCA substrate for flexible CMP cache sharing. In: Proc. of the 19th Intl. Conf. on Superc., pp. 31–40. ACM Press, New York (2005)
8. Iyer, R.: CQoS: a framework for enabling QoS in shared caches of CMP platforms. In: Proc. of the 18th Intl. Conf. on Superc., pp. 257–266. ACM Press, New York (2004)
9. Kim, S., Chandra, D., Solihin, Y.: Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In: Proc. of the 13th Intl. Conf. on Paral. Arch. and Comp. Techn., pp. 111–122. IEEE Computer Society, Los Alamitos (2004)
10. Liu, C., Sivasubramaniam, A., Kandemir, M.: Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs. In: Proc. of the 10th Intl. Symp. High Perf. Comp. Arch., p. 176. IEEE Computer Society, Los Alamitos (2004)
11. Mesa-Martinez, F.J., Nayfach-Battilan, J., Renau, J.: Power Model Validation Through Thermal Measurements. In: Proc. of the 34th Int. Symp. Comp. Arch., pp. 302–311. IEEE Computer Society, Los Alamitos (2007)
12. Speight, E., Shafi, H., Zhang, L., Rajamony, R.: Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors. In: Proc. of the 32th Int. Symp. Comp. Arch., pp. 346–356. IEEE Computer Society, Los Alamitos (2005)
13. SYNOPSIS, Inc., `http://www.synopsis.com`
14. Tarjan, D., Thoziyoor, S., Jouppi, N.P.: CACTI 4.0. HP Tech. Rep. Lab. HPL-2006-86 (2006)
15. UMC, `http://www.umc.com`
16. Zhang, C., Vahid, F., Najjar, W.: A Highly Configurable Cache Architecture for Embedded Systems. In: Proc. of the 30th Int. Symp. Comp. Arch., pp. 136–146. IEEE Computer Society, Los Alamitos (2003)

On-Chip COMA Cache-Coherence Protocol for Microgrids of Microthreaded Cores

Li Zhang[1] and Chris Jesshope[2]

[1] Informatics Institute, University of Amsterdam
[2] Kruislaan 403, Amsterdam 1098SJ, the Netherlands
{zhangli,jesshope}@science.uva.nl

Abstract. This paper describes an on-chip COMA cache coherency protocol to support the microthread model of concurrent program composition. The model gives a sound basis for building multi-core computers as it captures concurrency, abstracts communication and identifies resources, such as processor groups explicitly and where mapping and scheduling is performed dynamically. The result is a model where binary compatibility is guaranteed over arbitrary numbers of cores and where backward binary compatibility is also assured. We present the design of a memory system with relaxed synchronisation and consistency constraints that matches the characteristics of this model. We exploit an on-chip COMA organisation, which provides a flexible and transparent partitioning between processors and memory. This paper describes the coherency protocol and consistency model and describes work undertaken on the validation of the model and the development of a co-simulator to the Microgrid CMP emulator.

1 Introduction

It is now widely accepted that future computer systems must manage massive concurrency. Even on chip, that concurrency will be significant and asynchronous, having many of the same characteristics as grid systems. The constraints driving this are based on exponential functions meeting hard limits. For example, the hard limit on power dissipation will limit clock frequency and the limit of chip area is already a problem with respect to the area reachable in a single clock cycle. These constraints are the final nail in the coffin of the sequential model of computation. In the past, superscalar processors were able to exploit the implicit concurrency in sequential programs but they have very poor scaling as a consequence of their centralised model of synchronisation and scheduling. There is therefore a dire need for the development of more distributed models that still retain the advantages of the sequential model, namely composability and determinism.

Basic research into scalable, on-chip instruction execution has led in two distinct directions. The first has been the resurrection of dataflow instruction execution. In this model, instructions are mapped to ALUs and data is moved between instructions by using other instructions rather than named memory locations as targets of an operation [1,2]. The problem with this approach is that it does not support a memory model that can be used with conventional languages. Both approaches referenced have addressed this limitation but at some expense to the concurrency they are able to expose.

An alternative approach is to embrace explicit concurrency in the execution model, while at the same time maintaining the properties defined above that have made the sequential model so ubiquitous. The microthread model achieves this by capturing abstract concurrency in a conventional RISC-like ISA [3]. A *create* instruction dynamically defines concurrent execution as a *family* of threads based on a single thread definition. Because each member of the family has a unique index assigned to it, heterogeneous as well as homogeneous concurrency is supported. Families are parameterised and can be infinite in range. Concurrency can be created hierarchically as the thread definition for one family may contain creates for subordinate families. Create is therefore used to replace the sequential constructs of looping and function calls with concurrent equivalents. Moreover, the create instruction binds a *unit of work*, which is a family and its subordinate families to a *place*, which is a collection of processors.

Registers in this model implement a blocking read and provide fine-grain synchronisation between threads. This is similar to dataflow and in distinct contrast to other thread models such as Simultaneous Multi-Threading (SMT) [4] that synchronise on memory. The distributed register file also provides the mechanism for scheduling instructions from threads as continuations are stored in registers waiting for data, which are rescheduled on a write. The distribution of register files between multiple cores enables scalable data-driven execution of microthreaded code across many processors. It also provides significant tolerance to latency, as memory operations are decoupled by this register-based synchronisation. Current processor designs allow hundreds of threads to execute locally, typically allowing tolerance of up to a thousand cycles in memory accesses.

Threads in a family are mapped dynamically but deterministically to a set of cores and this provides the binary code compatibility. Moreover, legacy binary programs that use function calls and loops can be executed as singleton families in this model, providing backward compatibility.

While registers are used for fine-grain synchronisation between threads (and a thread and the memory system). A bulk synchronous model is provided on memory. Typically families of threads will update indexed data-structures in memory and a *sync* action (using a return code to a synchronising register) will provide the synchronisation required in order to create another family to consume the data structure. The memory model has a relaxed memory consistency compared to other thread-based approaches and requires the design of a new memory architecture and coherency protocol to fully exploit it. Memory written by a family is only defined following termination of all threads in that family. Race conditions are not excluded but the only reason they would exist (apart from bugs) is to allow explicit non-determinism, as may be found in some chaotic algorithms. We adopt location consistency on these races [5].

2 Background

With multi-core chips, it is even more difficult to break the 'memory wall' [6]. According to recent analysis (for instance Intel Pentium M platrom [7]), typical access to level 1 cache takes 1 to 3 cycles, access to level 2 cache takes around 10 to 20 cycles, while the RAM access may take more than a hundred cycles. Current trends in solving this problem are to increase the cache size and, as a result, the chip area of modern microprocessors is already dominated by cache. However a multi-core design must also consider scalable throughput from memory. An example is the IBM Power 4/5 architectures, which use three identical cache controllers for L2 cache, where cachelines are hashed across the controllers. Distribution brings further problems to the memory design and a choice must be made on how to implement sharing and coherence. In the Power 4/5 IBM provides coherence with four snoop processors implemented on the L2 controllers, whereas in their Cell processor they partition the memory locally and force the user to explicitly code the mapping of data to maintain coherence. Bus-based snooping on the other hand is not scalable.

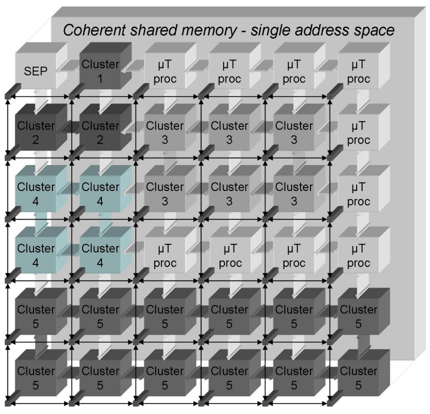
Trends in multi-cores designs can be seen in the Intel's 80-core Tera-Scale Research Chips [8], where each core has a local 256 KB memory associated via Through Silicon Vias (TSV) and where all processing cores are connected to the network on-chip. Such distributed structures provide scalability, but the local memory implementation lacks flexibility and would destroy the abstraction over mapping that gives binary code compatibility in the microthread model, where families of microthreads can be assigned freely to different processing cores on chip.

2.1 Requirements

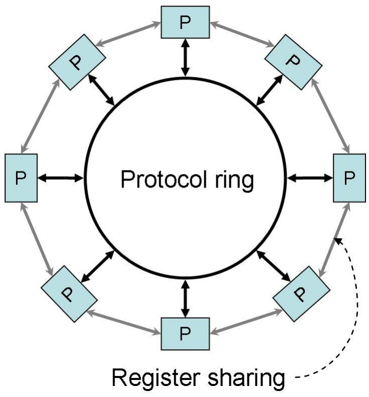
The requirement for a memory system in a Microgrid of microthreaded processors must provide the abstraction of a shared memory but achieve this across potentially thousands of processing cores, while providing scalable throughput both on and off chip. The ameliorating factor in this difficult design is that the processors tolerate a large amount of latency, which has led us to resurrect and specialise a paradigm used in earlier parallel computers, such as the Kendal Square KSR1 [9]. We introduce a Cache Only Memory Architecture (COMA) [10] for the on-chip cache system. In COMA, all the memory modules can be considered as large caches, called Attraction Memory (AM). Data is stored by cacheline but the line has no home location as a CC-NUMA [11], where the physical location of a memory address is always known. COMA adds complexity to locating a data in the memory but at the same time, increases the chances of data being in the local cache. In a Microgrid of microthreaded processors we propose a cache memory based on the COMA approach and allow data to migrate dynamically within the on-chip memory. A significant difference between the on-chip COMA and traditional COMA system is that the traditional COMA system will hold all data in the system without a backing store. However, on-chip COMA is unlikely to provide enough space to store so much data. The on-chip COMA therefore has a backing store for data off the chip, where one or more DRAM interfaces or links or some other Microgrid chips will provide an interface for storing incoming data. The main contribution of this paper is the protocol required to implement the memory consistency model in such an on-chip COMA memory system, its verification and subsequent use in a memory co-simulator.

2.2 The Microgrid Multi-core Chip

A Microgrid is a tiled multi-core chip designed using microthreaded multiprocessors. Because the microthread model uses a unique method of program composition using the create instruction, all structures in the microgrid must support this model. This includes on-chip networks and the memory organisation. At different levels in a program's concurrency hierarchy, families of threads are distributed to configured rings of processors, allocated from a pool of processors. This requires dynamic processor allocation similar to the way in which malloc is used for memory management. This is illustrated in Figure 1.a which shows a dynamic snapshot of an executing microthread program. The node marked SEP maintains a map of resources used and allocates and configures rings of processors over the low bandwidth grid network. The circuit-switched, ring networks that form clusters from the microthreaded processors provide protocols for family creation and termination and also allow adjacent processors in a ring to share data between their local register files to provide the distributed shared register file in the cluster. The ring network is shown in more detail in Figure 1.b.



(a)



(b)

Fig. 1. (a) Microgrid of microthreaded processors (μT proc) configured into dynamic clusters with ring interconnections. A request for a configured cluster is performed by the SEP over the resource management and delegation network, which is the low-bandwidth, packet-routed grid linking all processors. (b) Details of the ring network configured to support the execution of families of threads.

Each microthreaded processor in a cluster has a small (1KByte) level 1 D-cache. Missing in this cache does not stall the processor as memory operations are decoupled by suspending the thread on the target register location. Memory requests asynchronously update the register file when the memory access completes. In order to manage this, all memory requests must be tagged by target location in the register file as well as family identifier, for memory synchronisation.

The single address space defined in Figure 1.a is distributed over the chip in order to provide scalable performance. It is partitioned into Level 2 cache blocks, where a small number of processors will share requests on their L1 cache misses. In a traditional NUMA organisation, data has a home location, which means when the processor suffers a miss on the L2 cache, the processor will try to access the home location of the address. The access to a remote memory might take more than a thousand cycles. COMA, on the contrary, does not have a home location for a given piece of data. In COMA, the dynamic migration of data increases the possibility of finding the data in the local L2 cache or possibly another L2 cache module on chip. The downside of this is that to locate data is relatively expensive and to do this, directories are utilised. COMAs are also developed in different structures. Data Diffusion Memory (DDM) [12] uses a tree structure, where each level of the tree is associated with a Directory that holds information about the data availability below this level. (N.b. a directory does not store any data, just state information). The Kendall Square multiprocessor [9] on the other hand used a hierarchical snooping ring network. Generally, the protocols used are very similar to snooping protocols.

Details of the structure and protocol of the Microgrid on-chip COMA are given in the next section.

3 Memory Hierarchy Design

The memory system includes both the on-chip cache system and an off-chip communication interface or interfaces. Presently it is assumed that the off-chip interface is connected to a multi-bank memory storage, which is able to provide a high bandwidth for feeding multiple processors on a chip. The address space is physically interleaved across different memory banks.

The on-chip cache system is designed to have 2 levels of caches. Each processor is closely associated with its own small and fast L1 cache. L2 caches are relatively big, and each could supply data for multiple processor-L1 cache pairs. The L2 caches and their associated L1 caches are connected with snooping buses. Since the local bus configuration is not scalable, the number of L1 caches connected on a bus is restricted. Assuming each L2 cache can support between 4 to 8 processor-L1 cache pairs, then the number of L2 caches for current technology (say 128 processors) is 16 to 32. In future, perhaps thousands or even tens of thousands of processors may be integrated onto a single chip. Thus, the network utilised to connect L2 caches will be a hierarchy of ring networks, which map onto the hierarchical nature of the concurrency trees generated in the microthreaded model. We believe this will provide scalability in both bandwidth and cost as well as locality in communication.

To reduce the coherence pressure on a ring network, we group adjacent L2 caches together, and the caches in the same group are connected with a uni-directional ring network. Furthermore, all groups are connected by a higher level ring to allow for memory requests between groups. The lower and higher level ring networks are called level-1 (L1) and level-2 (L2) ring networks respectively. The joints between L1 and L2 rings are Directories, the structures designed to direct the flow of certain requests and reduce the network traffic. Each directory holds the information about all the data available in the group it is associated with. Like caches, directories hold information in a set-associative manner. Furthermore, the items in the directory are tailored to the cacheline size. Each item in the directory holds information about the cacheline tag and some state information without any data values. The state information can tell the availability and exclusiveness of a certain cacheline in the group it associates with. For instance, when a directory indicates a certain data is exclusive, it means that the certain cacheline can only be found valid in the current group, although the cacheline inside the group can be shared across different caches. On the L2 ring a Root Directory (RD) holds all the state information about the available data on-chip. It helps decide whether to send out a request off the chip. The memory controller connected with RD will carry out the off-chip communication.

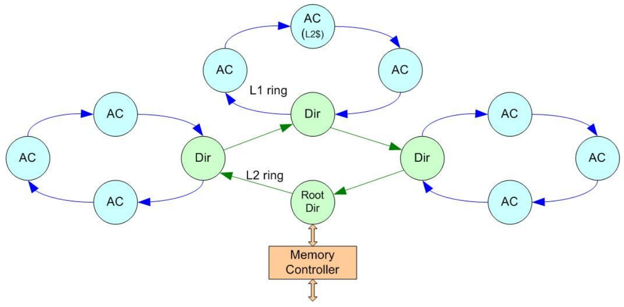


Fig. 2. Attraction Caches are organized around a 2-level hierarchical ring network as on-chip COMA

A normal snooping L2 cache only serves requests from processors and passively changes its state information. However, in this on-chip cache, each L2 cache not only serves the request received locally, but also serves requests received from the network. The property is very similar to COMA, which allows different Attraction Memories to serve data automatically so that data can flow to the place where it was mostly or recently used. Thus we call our on-chip cache system an on-chip COMA cache hierarchy. Furthermore, since the L2 cache behaves similarly to an attraction memory, to differentiate the cache from a normal cache we also call these L2 caches Attraction Caches (AC). The on-chip

COMA structure is depicted in Figure 2. In the figure, there are only 3 L1 rings and 3 ACs are shown, but the number of rings and ACs can be set arbitrarily. Since the memory access patterns in our model can be very different from normal processors due to its distributed large register file design and swift context swtich capability, the detailed design parameters have to be decided by co-simulation with our existing multi-processor simulator.

4 On-Chip COMA Cache-Coherence Protocol

A cache-coherence protocol maintains the consistency in a cache system. The design of the protocol for our model also has to address the issues such as minimising off-chip communication and providing a solution to the high bandwidth requirement. Our L1 cache is very small, simply providing a buffering and prefetching functionality. It uses a simple protocol with only two cacheline states, Valid and Invalid. Here we focus on the design of Attraction Cache protocol.

In a distributed shared memory architecture, almost all the cache coherence protocols are based on MOESI variations. MOESI represents five cacheline states, Modified (M), Owned (O), Exclusive (E), Shared (S), and Invalid (I) states. The cachelines in M and O states have the ownership of the data; they are also called dirty. The line at M/E states holds data exclusively. A shared cacheline only has the validness of data. Invalid data means the line is not present in the current cache. S and I states are the basic states that represent the validness of a cacheline. M and O states tend to keep the dirty values on the chip, which helps reduce off-chip communication. The Exclusive state is useful when repeated writes and reads to the same location happens in the cache. This situation is unlikely to happen because in our architecture data dependencies are generally captured at the register level rather than the memory level. Thus without Exclusive state a MOSI protocol is chosen. Since the memory requests are served asynchronously, the outstanding requests have to be remembered in the current cache. Consequently, two basic states are provided, ReadPending (RP) and WritePending (WP) states. The RP line waits for the valid data to be loaded and the WP line waits for the exclusiveness of the line to be acquired before performing a Write operation locally.

The AC is able to handle 10 different requests, which are listed on Figure 3. In the following text only the acronyms are mentioned. Requests LR and LW can only be issued by the processor. Both RS and RE try to load the data remotely, while RE will acquire exclusiveness at the same time. SR and ER are the corresponding replies for RS and RE. The request BR represents the eviction of a cacheline. The BR can invalidate a shared line directly; it also has to preserve the dirty data by writing them back to the main memory with WB request. The IV request is normally generated by an LW request, which needs to acquire the exclusiveness of a data copy. When the IV returns to the initiator, it is regarded as a DE which represents the acquisition of the data exclusiveness.

As described above, the directories hold information about the current state of the data in the group. Three different states, Invalid (IN), Shared (SH), and

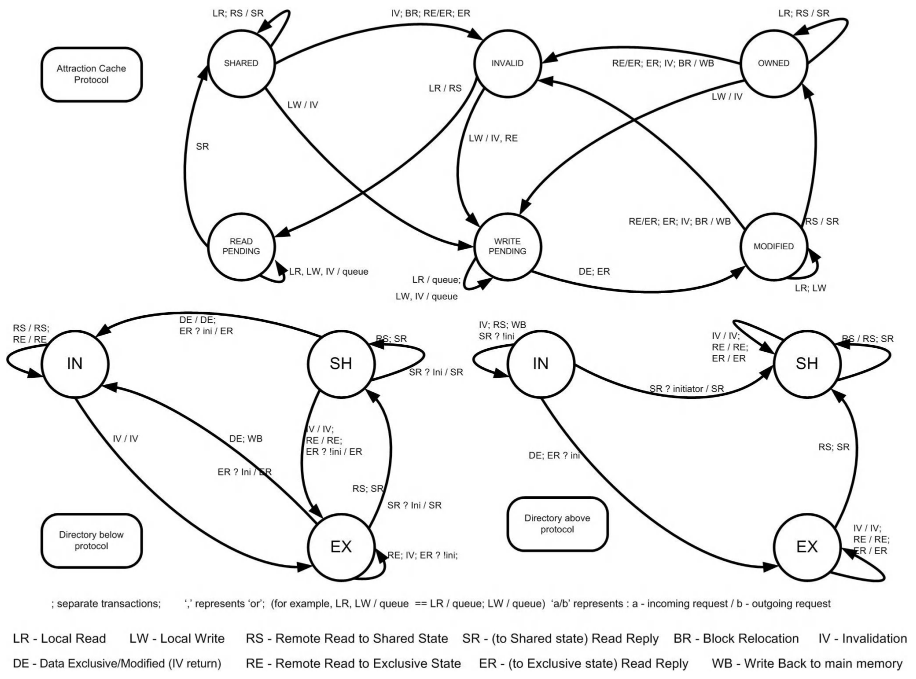


Fig. 3. MOSI protocol State Transition Diagram of Attraction Cache and Directory

Exclusive (EX), can be assigned to each item in a directory. As a joint of L1 and L2 rings, a directory also determines whether a certain request should be passed to the next node in the same or a different level. For instance, an RE request is received from L1 ring by a directory which has the corresponding item at EX state. Being aware the sub-system holds the exclusiveness, the directory will propagate the RE in the L1 level without incuring any traffic on L2 ring. The detailed state transitions of AC and directory are depicted in Figure 3.

5 Consistency Model

In a multi-processor system, a consistency model places specific requirement on the sequence that shared memory accesses from one process may be observed by other processes. A number of consistency models have been proposed, including Sequential Consistency (SC) [13], Release Consistency (RC) [14], and Location Consistency (LC) [5]. Different consistency models balance the programming complexity and system performance. LC is claimed to be the weakest consistency model. Unlike SC, LC does not make the assumption that all writes to the same memory location are serialized in some order observable to all processors. The program will behave the same as on other systems as deterministic code is being executed. Furthermore, unlike RC the synchronization is not happening for different blocks of code, but in terms of each individual memory location. This partial order is only maintained for each individual memory locations. The issuing order between memory accesses on different memory locations can be adjusted by the compiler to achieve better performance.

In our on-chip COMA architecture, the maintenance of sequential consistency is very expensive. For instance, if two processors are writing to the same memory location concurrently on a ring. The processors separate the ring into two arcs. The ACs on different arcs will observe the two requests in different orders, which is forbidden in the sequential consistency model. Fortunately the microthread memory model allows us to adopt the more relaxed LC as the consistency model, which does not require the strict order under non-deterministic situations.

To exploit the potential of Location Consistency, a suspended request queue structure is proposed for each cache and directory. In the Attraction Cache, a cacheline will be locked in a temporary state when a new request to the same location cannot be served directly. Thus, the incoming request has to be saved temporarily in a buffer. To avoid blocking the request to other memory locations which can be served directly, a queue structure is proposed to be associated with each suspended cacheline. As the reply to the locked line returns, the associated suspended line can be reactivated and served directly. Since we are using the LC consistency model, the order of serving requests from different locations does not need to be handled by the memory system. As a result, LC is actually extended to the cacheline level. The temporary ReadPending and WritePending states are actually combinations of the states in the data table and Tstates in queue table depicted in Figure 4. Each queue head in the queue table is associated with a linked list in the request buffer. An Empty Queue Head (EQH) maintains the list of empty slots in the request buffer. Similar queue structure is also implemented in the directories.

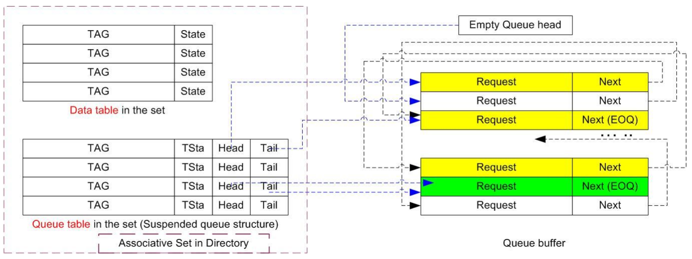


Fig. 4. Suspended Queue Structure in Attraction Cache

6 Conclusion and Future Work

The paper gives an overview of the microthreaded CMP architecture. With its capability for explicit context switching and scheduling based on thread continuations held in a large distributed register files, the model can tolerate long memory access latency and give high throughput. Targeting VLSI technology in the 10 to 15 year timeframe, we have introduced an on-chip distributed shared memory architecture and defined its operation. The choice of an on-chip COMA system is to more efficiently utilise the overall memory bandwidth. Two levels of ring networks are utilised to distribute memory storage across the network for a large number of processors on chip and directories are used to direct the memory transaction flow within the on-chip cache hierarchy. We have analised the microthreaded memory model and have used Location Consistency as the consistency model. Finally, for coherence a variant of the MOSI protocol has been specified and implemented to maintain coherence for the caches. A novel feature of this work is the proposed Suspended Request Queue implementation for both Attraction Cache and Directory to further reduced the traffic on the network.

Currently we are intensively verifying the cache coherence protocol by specifying its complete behavior in Murphy description language [15]. The language allows the user to specify initial system state and rules in addition to the procedures. From the initial state Murphi will automatically fire different rules according to the conditions specified for them. The process will continue until all system states are explored. By checking the correctness of each system state, the protocol can be verified. The technique is called State Enumeration [16]. At the current stage, the protocol has been proved free of deadlock with Murphi and we are verifying the implementation of location consistency.

In addition, the simulation of the memory system using SystemC has been completed and tested using synthetic traces. In the near future, this memory simulator will be integrated into the Microgrid CMP emulator to evaluate the overall chip architecture in a cycle-accurate manner. The design parameters and protocols will then be be tuned for the on-chip COMA memory system. Furthermore, we are developing a memory protection scheme that will provide families of microthreads exclusive access to memory protection domains.

References

1. Mercaldi, M., Swanson, S., Petersen, A., Putnam, A., Schwerin, A., Oskin, M., Eggers, S.J.: Instruction scheduling for a tiled dataflow architecture. In: ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, pp. 141–150. ACM Press, New York (2006)
2. Burger, D., Keckler, S.W., McKinley, K.S., Dahlin, M., John, L.K., Lin, C., Moore, C.R., Burrill, J., McDonald, R.G., Yoder, W.: The TRIPS Team: Scaling to the end of silicon with edge architectures. Computer 37(7), 44–55 (2004)
3. Jesshope, C.R.: A model for the design and programming of multicores. In: Advances in Parallel Computing, IOS Press, Amsterdam (Draft paper submitted, 2007)
4. Tullsen, D.M., Lo, J.L., Eggers, S.J., Levy, H.M.: Supporting fine-grained synchronization on a simultaneous multithreading processor. In: HPCA 1999: Proceedings of the 5th International Symposium on High Performance Computer Architecture, Washington, DC, USA, p. 54 (1999)

5. Gao, G.R., Sarkar, V.: Location consistency-a new memory model and cache consistency protocol. IEEE Transactions on Computers 49(8), 798–813 (2000)
6. Wulf, W.A., McKee, S.A.: Hitting the memory wall: implications of the obvious. SIGARCH Comput. Archit. News 23(1), 20–24 (1995)
7. Besedin, D.: Testing platforms with rightmark memory analyzer benchmark. part 5: Intel pentium m platform (dothan) (May 2004), `http://www.digit-life.com/articles2/rmma/rmma-dothan.html`
8. Intel Corporation: Intel develops tera-scale research chips (September 2006), `http://www.intel.com/go/terascale/`
9. Corporation, K.S.R.: Ksr1 technical summary. Technical report (1992)
10. Dahlgren, F., Torrellas, J.: Cache-only memory architectures. Computer 32(6), 72–79 (1999)
11. LeBlanc, T.J., Marsh, B.D., Scott, M.L.: Memory management for large-scale numa multiprocessors. Technical report, Rochester, NY, USA (1989)
12. Hagersten, E., Landin, A., Haridi, S.: DDM - a cache-only memory architecture. IEEE Computer 25(9), 44–54 (1992)
13. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Computers 28(9), 690–691 (1979)
14. Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P.B., Gupta, A., Hennessy, J.L.: Memory consistency and event ordering in scalable shared-memory multiprocessors. In: 25 Years ISCA: Retrospectives and Reprints, pp. 376–387 (1998)
15. Dill, D.L.: The murphi verification system. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 390–393. Springer, Heidelberg (1996)
16. Pong, F., Dubois, M.: Verification techniques for cache coherence protocols. ACM Computing Surveys 29(1), 82–126 (1997)

Parallelization of Bulk Operations for STL Dictionaries

Leonor Frias[1,⋆] and Johannes Singler[2]

[1] Dep. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya
[2] Institut für Theoretische Informatik, Universität Karlsruhe
`lfrias@lsi.upc.edu, singler@ira.uka.de`

Abstract. STL dictionaries like `map` and `set` are commonly used in C++ programs. We consider parallelizing two of their bulk operations, namely the construction from many elements, and the insertion of many elements at a time. Practical algorithms are proposed for these tasks. The implementation is completely generic and engineered to provide best performance for the variety of possible input characteristics. It features transparent integration into the STL. This can make programs profit in an easy way from multi-core processing power. The performance measurements show the practical usefulness on real-world multi-core machines with up to eight cores.

1 Introduction

Multi-core processors bring parallel computing power to the customer at virtually no cost. Where automatic parallelization fails and OpenMP loop parallelization primitives are not strong enough, parallelized algorithms from a library are a sensible choice. Our group pursues this goal with the Multi-Core Standard Template Library [7], a parallel implementation of the C++ STL. To allow best benefit from the parallel computing power, as many operations as possible need to be parallelized. Sequential parts could otherwise severely limit the achievable speedup, according to Amdahl's law. Thus, it may be profitable to parallelize an operation even if the speedup is considerably less than the number of threads.

The STL contains four kinds of generic dictionary types, namely `set`, `map`, `multiset`, and `multimap`. The first two are unique, i. e. no two equal elements may be included. Parallelization of the bulk operations of these containers has been one of the major missing parts of the MCSTL, so far. Now, we have implemented all variants, but limit our description to `set` in this paper, as it is one of the more general and more complicated ones, at the same time. The fine-grained basic operations, i. e. insertion, query and deletion of a single element, are not worth parallelizing because parallelization overhead would dominate their logarithmic running time. But often, many elements are inserted at a time, or a

⋆ Supported by ALINEX project (TIN2005-05446) and by grants number 2005FI 00856 and 2006BE-2 0016 of the *Agència de Gestió d'Ajuts Universitaris i de Recerca* with funds of the European Social Fund. This work was partially done while visiting Universität Karlsruhe.

dictionary is constructed newly and initialized by a large sets of elements. Of course, the latter is a special case of the former, namely inserting into an empty dictionary. Nevertheless, we treat both cases explicitly, for best performance.

Our parallelization augments the STL implementation coming with the GNU C++ compiler (libstdc++), which uses red-black trees [4]. Red-black trees are balanced binary trees[1] whose nodes can be either colored red or black, the root always being black. The following invariant is maintained at all times: Each path from the root to a leaf must contain the same number of black nodes. This limits the tree height to $2\lceil \log n \rceil$. By performing so-called *rotations*, the invariant can be held up efficiently for all operations.

2 Algorithms

For the PRAM model, parallel red-black tree algorithms have been proposed [5]. They are highly theoretical, use fine-grained pipelining etc., and are thus not suitable for real-world machines. To our knowledge, there is neither an implementation nor experimental results. Nonetheless, some of the high-level ideas can be transferred to practical algorithms.

The STAPL [1] library does provide a parallel tree implementation, with an interface similar to the STL but it is not publicly available. It also works for distributed-memory systems, but makes a completely new implementation necessary. Its quite rigid partitioning of the tree can lead to all the elements being inserted by one processor, in the worst case. In contrast, we have implemented our bulk insertion operations on the top of the sequential data structure core, which stays unaffected. Our partitioning of the data is temporary for one operation and more flexible, e. g. multiple threads can work on a relatively small part of the tree. The cost of splitting the tree is negligible compared to the cost of creating/inserting the elements into the tree, for the common case.

As stated before, bulk construction can be seen as a special case of insertion. Indeed, we treat it the other way around here. Bulk construction (of a subtree) is the base case for bulk insertion. This way, we get good performance also for the case where many elements are inserted between two already contained elements (or conceptually, $-\infty$ and ∞ for bulk construction).

Before describing the parallel tree construction and the bulk insertion, we present the common setup for both, i. e. preprocessing the data to enable parallelization, and allocating node objects. Let p be the number of threads used, n the size of existing tree, and k the number of elements to insert and to construct the tree from, respectively.

Common Setup. The first step is to make sure the input sequence S is sorted, which also is a precondition in [5]. If two wrongly ordered elements are found, checking the order is aborted, and the input sequence is sorted stably, using the existing MCSTL implementations. Because the actual input sequence must not

[1] Note that hash data structures are not feasible here since STL dictionaries support sorted sequence functionality as well.

be modified, we need linear temporary storage here. Finally, S is divided into p subsequences of (almost) equal size.

Allocation and Initialization. Each thread t allocates and constructs the nodes for its subsequence S_t, which are still unlinked. The tree nodes are stored in an array of pointers, shared among the threads. This allows the algorithm to easily locate the nodes for linking, later on. If we wanted to avoid the array of pointers, we would need a purely recursive algorithm where allocation and initialization of the pointers are intertwined.

We also deal with equal elements in this step. In parallel, the surplus elements are removed from the sequence. A gap might emerge at the end of each S_t. Since we use stable sorting, we can guarantee that the first equal element becomes the one inserted, as demanded by the specification.

2.1 Parallel Tree Construction

Once the common setup has been performed, the tree is constructed as a whole by setting the pointers and the color of each node, in parallel. For each node, its pointers can be calculated independently, using index arithmetic. This takes into account the gaps stemming from removed equal elements, as well as the incompleteness of the tree. Each node is only written to by one thread, there is no need for synchronization. Thus, this is perfectly parallelized.

The algorithm constructs a tree that is complete except for the last level, which is filled partially from left to right. The last level is colored red, all other nodes are colored black.[2] Another option would be to also color every ℓ^{th} level red. The smaller ℓ, the more favored are deletions in the subsequent usage of the dictionary. The larger ℓ, the more favored are insertions.

2.2 Parallel Bulk Insertion

The problem of inserting elements into an existing tree is much more involved than construction. The major question is how to achieve a good load balance between the threads. There are essentially two ways to divide the work into input subsequences S_t and corresponding subtrees T_t. *1.* We divide the tree according to the input sequence, i. e. we take elements from the sequence and split the tree into corresponding subtrees. *2.* We divide the input sequence according to the tree, i. e. we split S_t by the current root element, recursively. Both approaches fail if very many elements are inserted at the end of the path to the same leaf. The first approach gives guarantees only on $|S_t|$, but not on $|T_t|$. The second approach does not guarantee anything about $|S_t|$, and bounds $|T_t|$ only very weakly: One of the subtrees might have twice the height as the other, so only $|T_1| < |T_2|^2$ holds. We use the first option in the initial step, and the second approach to compensate for greatly different $|T_t|$ dynamically.

[2] We could omit this if the tree is complete, but this is a rare special case. For a tree containing only one element, the root node is colored black anyway.

The split and concatenate operations on red-black trees [8,9] are the major tools in our parallelization of the bulk insertion operation.

Split into Multiple Subtrees. A red-black tree is split into p red-black trees, such that the elements will be distributed according to $p-1$ pivots.

Concatenate. Two range-disjoint red-black trees and a node "in-between" are concatenated to form a single red-black tree. Rebalancing might be necessary.

The whole algorithm consists of four phases. For coordinating the work, *insertion tasks* and *concatenation tasks* are generated, to be processed in the future.

Phase 0. Common setup, as described above.

Phase 1. Split the tree into p subtrees, the leftmost elements of the subsequences (except the first one) acting as splitters. This is performed in a top-down fashion, traversing at most $p-1$ paths, and generating $p-1$ concatenation tasks. With each concatenation task, we associate a node which is greater than all elements in the right subtree, but smaller than all elements in the right subtree. The algorithm chooses either the greatest element from the left subtree, or the least element from the subsequence to insert. This node will be used as tentative new root when concatenating the subtrees again, but might end up in another place, due to rebalancing rotations. The resulting subtrees and the corresponding subsequences form p independent insertion tasks, one for each thread.

Phase 2. Process the insertion tasks. Each thread inserts the elements of its subsequence into its subtree, using an advanced sequential bulk insertion algorithm (see Section 2.4).

Phase 3. Process the concatenation tasks to rebuild and rebalance the tree. This phase is not actually temporally separated from Phase 2, but only conceptually. As soon as one thread has finished processing an insertion task, it tries to process its parent concatenation task, recursively. However, this can only happen if the sibling subtask is also already done. Otherwise, the thread quits from this task and continues with another. It takes that one from its local queue or steals from another thread, if the own queue is empty. The root concatenation task does not have any parent, so the algorithm terminates after having processed it.

2.3 Analysis

We analyse the parallel time complexity of our algorithms. Assume for simplicity that the input sequence has already been preprocessed. To get rid of lower-order terms, we assume the number of elements in the tree being relatively large with respect to the number of threads, $n > p^2$.

Our construction algorithm takes $O(k/p)$ parallel time.

Calculating the worst-case cost for the bulk insertion algorithm is more involved. Splitting the tree sequentially into p parts takes $O(p \log n)$ time. As tests have shown, performing this step in parallel is not helpful assuming the number of cores currently available. In the worst case, one of the partitions contains a tree of size (almost) n, and the others contain (almost) empty trees.

Inserting a subsequence of size k/p sequentially into a tree of constant size takes $O(k/p)$ time. Inserting a sequence of size k/p sequentially into a tree of size n is upper bounded by the cost of inserting the elements one by one, i. e. $O(k/p \log n)$. Therefore, doing k insertions in p disjoint trees takes $O(k/p \log n)$ parallel time.

Finally, the p trees must be concatenated. Note that the concatenation operation itself is done sequentially but concatenations of disjoint trees will happen in parallel. A concatenation takes $O(\log n_1/n_2)$ sequential time, where n_1 is the size of the larger tree and n_2 is the size of the smaller subtree. Besides, the trees to be concatenated can differ in size by at most n elements. Therefore, one concatenation takes at most $O(\log n)$ time. Given that there are $O(\log p)$ levels of concatenations in total, the cost of concatenating all the trees is $O(\log p \log n)$.

As a result, the total cost of the operation is dominated by the insertion itself and therefore, it takes $O(k/p \log n)$ parallel time.

2.4 Dynamic Load-Balancing

The sequence of the elements to insert is perfectly divided among the threads. However, the corresponding subtrees might have very different sizes, which of course affects (wall-clock) running time negatively. Even worse, it is impossible to predict how elaborate the insertion will be, since this depends on the structure of the tree and the sequence. To counter this problem, we add dynamic load-balancing to Phase 2, using the randomized work-stealing [3] paradigm.

Each thread breaks down its principal insertion task into smaller ones. It goes down the subtree and divides the subsequence recursively, with respect to the current root. For each division, it creates the appropriate concatenation task, in order to reestablish the tree in Phase 3. The insertion task for the right subtree is pushed to the thread's work-stealing queue, while the recursion continues on the left subtree. However, the queue is only used if S_t is still longer than a threshold s.Otherwise, the algorithm works on the problem in isolation, to avoid the overhead of maintaining the queue. When there is nothing left to split, the bulk construction method is called for the remaining elements, and the new tree is concatenated to the leaf. If the subsequence has only a few elements left, the single-element insertion algorithm is called. As a side-effect, this gives us an more efficient sequential bulk-insertion algorithm, for s conceptually set to ∞.

To implement work-stealing, we use the lock-free double-ended queue provided by the MCSTL. It allows efficient synchronization of the work, using hardware-supported atomic operations. On the downside, its size must be limited at construction time. Fortunately, the size of all queues is bounded by the height of the tree, namely $2\lceil \log n \rceil$. In total, the approach is similar to the parallelized quicksort in the MCSTL [7, p. 6].

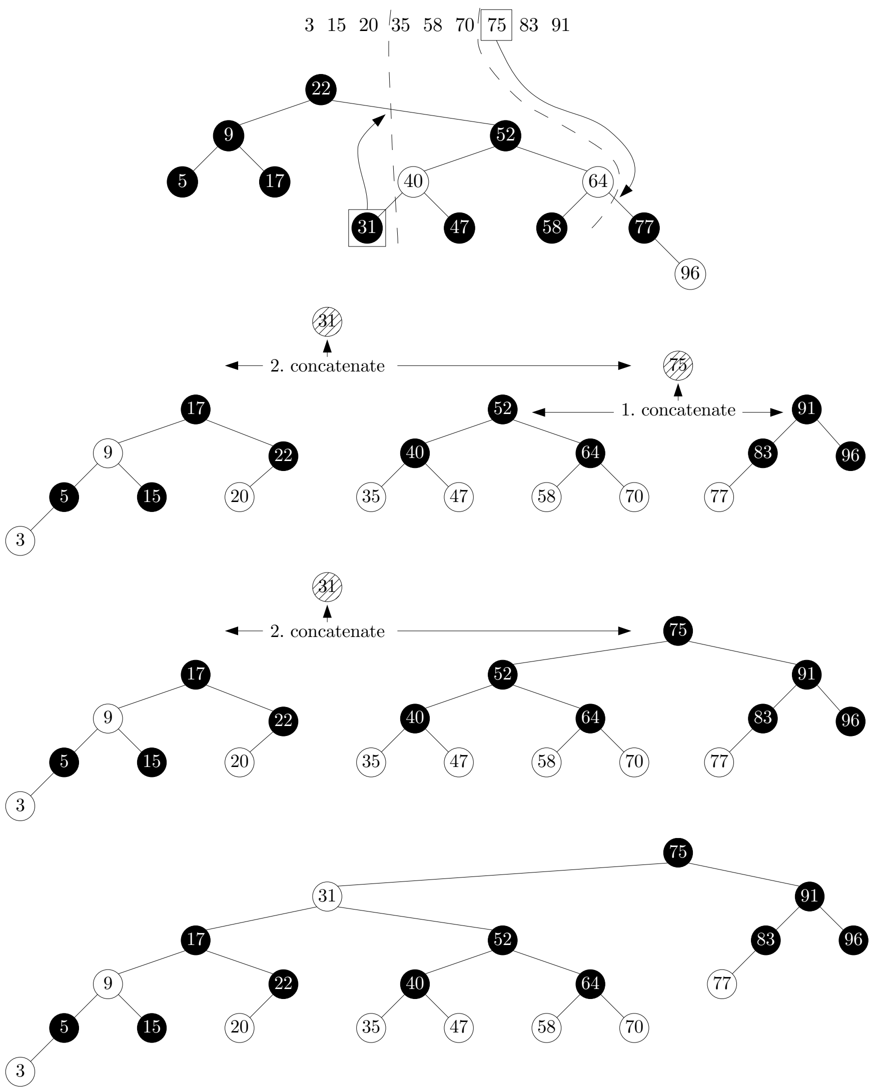


Fig. 1. Inserting into a tree, using three threads. "Red" nodes are shown transparent. *1.* Phase 0/1. The input sequence is divided equally, the tree is split accordingly. *2.* Phase 2. The two elements that will serve as the tentative roots in the future concatenation tasks are found to be 31 and 75. They are excluded from the subtree and not inserted to it, respectively. 58 is not inserted either, since it is a duplicate. *3./4.* Phase 3. Eventually, the concatenation task are processed. First, the right two subtrees are concatenated, using 75 as the new root. Then, the resulting tree is concatenated with the left one using 31 as the new root. In this case, a rotation is necessary for rebalancing purposes, so 75 ends up at the top position.

3 Interface and Implementation Aspects

Memory Management. For both bulk operations, allocating the node objects constitutes a considerable share of the work to be done. We cannot allocate several of them with one function call, since we need to be able to deallocate them one by one in case the user deletes elements from the data structure. The memory management concept of C++ does not allow such an asymmetric usage. Tests have shown that allocating memory concurrently scales quite well, but imposes some work overhead compared to the strictly sequential implementation. There also exist memory managers designed for this specific problem, e. g. Hoard [2]. We successfully used it on the Sun machine. However, for our 64-bit Intel platform, Hoard could not even match the performance of the built-in allocator, so we did not use it there.

Trade-Offs for Different Input Data and Comparator Types. STL dictionaries, apart from being parameterized by the data type, are also customized by a *comparator* type. This can lead to very different costs for comparing and copying an element. We also have to relate this to the cost of copying the node as a whole, which contains three pointers and the node color flag, in addition to the actual payload.

We tested this for sorting the input elements, experimentally. For large elements, it is better to sort pointers only, to avoid costly copying. For small elements, it is more cache-efficient to sort the elements directly.

4 Experimental Results

Experiments. We tested our program on two multi-processor machines with the following processors: *1.* Sun T1 (1 socket, 8 cores, 1.0 GHz, 32 threads, 3 MB shared L2 cache), *2.* Intel Xeon E5345 (2 sockets, 2×4 cores, 2.33 GHz, $2 \times 2 \times 4$ MB L2 cache, shared among two cores each). For the Intel machine, we used the Intel C++ compiler 10.0.25, on the Sun machine, we took GCC 4.2.0, always compiling with optimization (`-O3`) and OpenMP support. In both cases, we used the libstdc++ implementation coming with GCC 4.2.0. On the Xeon, compiling using the Intel compiler lead to more consistent results, due to the better OpenMP implementation, which is causing less overhead.

We have run each test at least 30 times and taken the average values for the plots, accompanied by the standard deviation range (too small to be seen in most cases). The following parameters concerning the input were considered:

Tree size/insertion sequence length ratio. Let n be the size of the existing dictionary/tree. Let $r = n/k$. Thus, $r = 0$ is equivalent to bulk construction.

Sortedness of the insertion sequence. The input sequence can be presorted or not. We focus on presorted insertion sequences here because parallel sorting is a separate issue.

Key data type. We show experiments on 32-bit signed integers.

Randomness of the input sequence. The input sequence by default consists of values in the range {`RAND_MIN`...`RAND_MAX`}, but can optionally be limited to a smaller range, {`RAND_MIN`/100 ... `RAND_MAX`/100} here. This poses a challenge to the load-balancing mechanisms.

Both load-balancing mechanisms were switched on by default, but deactivated for some experiments, to show their influence on the running time.

For the insertion tests, we first build a tree of the appropriate size using the sequential algorithm, and then insert the elements using our parallel bulk insertion. This guarantees fair initial conditions for comparing the sequential and the parallel implementation. Actually, the shape of the existing tree does affect performance, though the sequential insertion is more affected than our algorithms.

Results for the Sequential Case. Our sequential algorithm is never substantially slower and in some cases substantially faster (e. g. Figures 2, 4 and 5). In particular, our specialized algorithms are very competitive when having random inputs limited to a certain range, being more than twice as fast (see Figures 8 and 9) as the standard implementation. This is because of saving many comparisons in the upper levels of the tree, taking advantage of the input characteristics.

Results for the Parallel Case. For the dictionary construction, our algorithms scale quite well, as shown in Figures 2 and 3. In particular, on the 8-core Sun T1, even the absolute speedup slightly exceeds the number of cores, culminating in about 11. This shows the usefulness of per-core multithreading in this case.

For insertion, our algorithm is most effective when the existing tree is smaller than the data to insert (see Figures 5 and 6). This is also due to the fast (sequential) insertion procedure, compared to the original algorithm. Splitting the input sequence is most effective in terms of number of comparisons because the subsequences left when reaching a leaf still consist of several elements.

In all cases, speedups of at least 3 for 4 threads and 5 for 8 threads, are achieved. The break-even is already reached for as little as 1000 elements, the maximum speedup is usually hit for 100000 or more elements.

The tree splitting step, used to make an initial balancing in the case of insertion, is shown to be really effective. For instance, compare Figures 6 and 7, which only differ in whether Phase 1 of the algorithm is run. We can see that both the speedup and the scalability are far better when the initial splitting is activated.

On top of that, the parallelization scales nicely, specifically when using dynamic load-balancing. Switching load-balancing off hurts performance for large inputs, while for small inputs it does not create considerable overhead. Dynamic load-balancing makes the algorithm more robust, comparing Figures 8 and 9.

As a by-product, we show the effects of mapping 2 threads to cores in different ways (see Figure 4). For small input, the most profitable configuration is sharing the cache, i. e. both threads running on the same die. As the input data

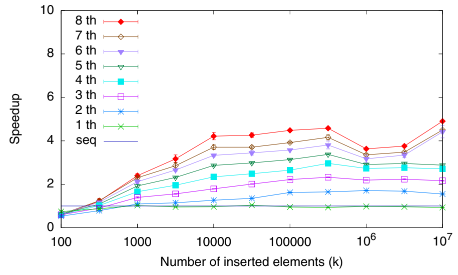


Fig. 2. Constructing a set of integers on the Xeon

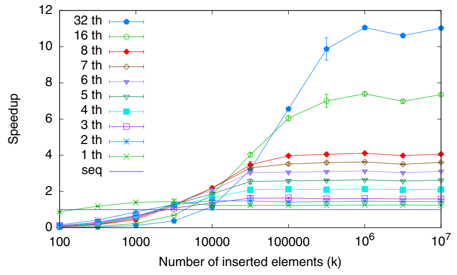


Fig. 3. Constructing a set of integers on the T1

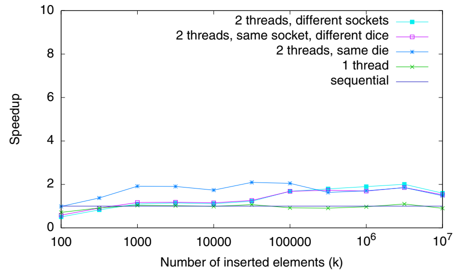


Fig. 4. Constructing a set of integers on the Xeon fixing two cores

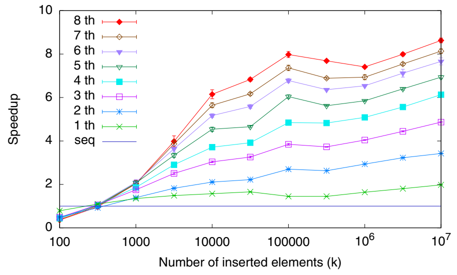


Fig. 5. Inserting integers into a set ($r = 0.1$) on the Xeon

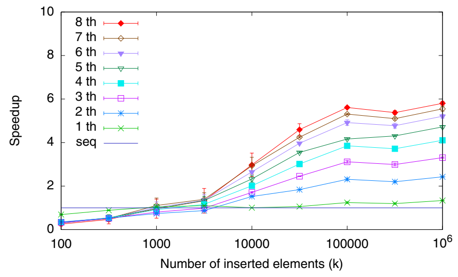


Fig. 6. Inserting integers into a set ($r = 10$) on the Xeon

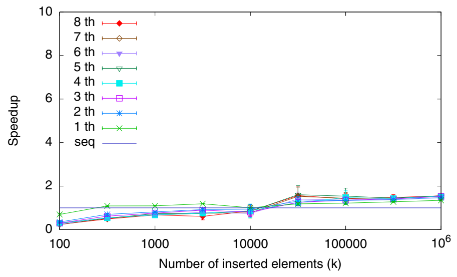


Fig. 7. Inserting integers into a set ($r = 10$) on the Xeon, without using initial splitting of the tree

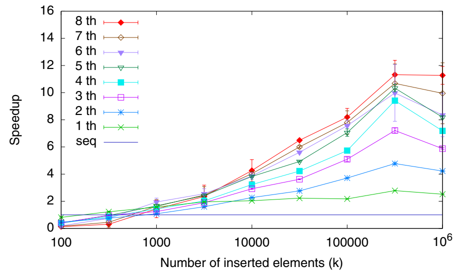


Fig. 8. Inserting integers from a limited range into a set ($r = 10$) on the Xeon

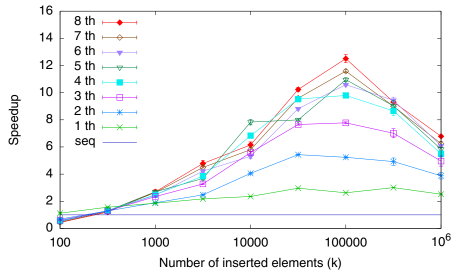


Fig. 9. Inserting integers from a limited range into a set ($r = 10$) on the Xeon, without using dynamic load-balancing

gets larger, memory bandwith becomes more important, so running the threads on different sockets wins. Running both threads on the same socket, but on different dice is in fact worst, because it combines both drawbacks. For the other tests, we have refrained to explicitly bind threads to specific cores, however, because the algorithms should be allowed to run on a non-exclusive machine in general.

5 Conclusion and Future Work

In this paper, we have shown that construction of and bulk insertion into a red-black tree can be effectively parallelized on multi-core machines, on top of a sequential implementation which remains unaffected. Our construction bulk operation shares some ideas with the theoretical algorithm in [5], giving a practical implementation. The code has been released in the MCSTL [6], version 0.8.0-beta.

To speed up programs that do not use bulk operations explicitly, we could use lazy updating. Each sequence of insertions, queries, and deletions could be split into consecutive subsequences of maximal length, consisting of only one of the operations. This approach could transparently be covered by the library methods. Then, for example, a loop inserting a single element in each iteration, would also benefit from the parallelized bulk insertion.

Another enhancement to the functionality is bulk *deletion* of elements. Since it is easy to remove a consecutive range from a tree, we will rather tackle the problem posed by a `remove_if` call, specializing it for the dictionary types.

References

1. An, P., Jula, A., Rus, S., Saunders, S., Smith, T., Tanase, G., Thomas, N., Amato, N.M., Rauchwerger, L.: STAPL: An Adaptive, Generic Parallel C++ Library. In: LCPC, pp. 193–208 (2001), `http://parasol.tamu.edu/groups/rwergergroup/research/stapl/`
2. Berger, E.D., McKinley, K.S., Blumofe, R.D., Wilson, P.R.: Hoard: A scalable memory allocator for multithreaded applications. In: ASPLOS-IX (2000)
3. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. JACM 46(5), 720–748 (1999)
4. Guibas, Sedgewick: A dichromatic framework for balanced trees. In: FOCS (1978)
5. Park, H., Park, K.: Parallel algorithms for red-black trees. Theoretical Computer Science 262, 415–435 (2001)
6. Singler, J.: The MCSTL website (June 2006), `http://algo2.iti.uni-karlsruhe.de/singler/mcstl/`
7. Singler, J., Sanders, P., Putze, F.: The Multi-Core Standard Template Library. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, Springer, Heidelberg (2007)
8. Tarjan, R.E.: Data structures and network algorithms. In: CBMS-NSF Regional Conference Series on Applied Mathematics, vol. 44, SIAM, Philadelphia (1983)
9. Wein, R.: Efficient implementation of red-black trees with split and catenate operations. Technical report, Tel-Aviv University (2005)

UNICORE Summit 2007
(Foreword)

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" during October 11–12, 2005 in Sophia Antipolis, France. In 2006 the style of the UNICORE Summit was changed by establishing a Program Committee and publishing a Call for Papers. The UNICORE Summit 2006 was held in conjunction with the Euro-Par 2006 conference in Dresden, Germany, August 30–31, 2006. The proceedings are available as LNCS 4375. In 2007 the UNICORE Summit was held again in conjunction with the Euro-Par conference, this time in Rennes, France, on August 28.

We would like to thank the Program Committee members Agnes Ansari, Rosa Badia, Thomas Fahringer, Donal Fellows, Anton Fank, Edgar Gabriel, Alfred Geiger, Odej Kao, Paolo Malfetti, Ralf Ratering, Johannes Reetz, Mathilde Romberg, Bernd Schuller, Dave Snelling, Stefan Wesner, and Ramin Yahyapour for their excellent job. Special thanks go to Sven van den Berghe and Morris Riedel for providing additional reviews.

Finally, we would like to thank all authors for their submissions, camera-ready versions, and presentations at the UNICORE Summit 2007 in Rennes as well as Ken Klingenstein for giving the opening talk.

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

November 2007

Achim Streit
Wolfgang Ziegler

A Black-Box Approach to Performance Analysis of Grid Middleware

Per Alexius, B. Maryam Elahi, Fredrik Hedman, Phillip Mucci, Gilbert Netzer, and Zeeshan Ali Shah

Center for Parallel Computers (PDC), KTH, SE-100 44 STOCKHOLM, Sweden
{alexius,elahi,hedman,mucci,noname,zashah}@kth.se

Abstract. We propose a black-box approach to performance analysis of grid middleware and present the architecture of a non-invasive platform-independent evaluation tool that quantifies the effects of the overhead imposed by grid middleware on both the throughput of the system and on the turnaround times of grid applications. This approach is a step towards producing a middleware independent, comparable, reproducible and fair performance analysis of grid middlewares. The result of such performance analysis can be used by system administrators to tune the system configuration and by developers to find the bottlenecks and problems in their design and implementation of the system. It can also be used to devise more optimized usage patterns. As a proof of concept, we describe the implementation details of the evaluation tool for UNICORE 5 and demonstrate the result of initial experiments.

Keywords: UNICORE, Performance Analysis, Grid Middleware.

1 Introduction

Performance analysis and benchmarking are relatively young and emerging areas in Grid computing which itself is only about a decade old. Grids are comprised of a hierarchy of heterogeneous resources that are managed by several interacting software components [1]. This inherent complexity frustrates the direct application of some of well established techniques in parallel or distributed computing, among them performance analysis and benchmarking [2].

A few research groups have presented their interpretation of what the parameters of interest in grid performance assessment are and have also proposed methods and developed tools for evaluation of grid systems [2,3,4,5]. However, benchmarking and performance analysis of grid middlewares are a less explored area of research. The characteristic of grid applications is that they vary in resource requirements. The resource that is ultimately allocated must satisfy the user specified minimum requirements, but could be very different from one run to another. This makes platform-centric performance assessment of any particular application of limited use to grid developers, and grid users. Although resource benchmarking could be used as a tool for assisting better resource ranking and allocation [6], it can hardly be used to reflect on the performance of the grid middleware. This is for two reasons. The first is that platform-centric metrics places undue restrictions on the underlying execution platform to provide hardware and software components able to provide these specific measurement capabilities; also these components can not be guaranteed to be available. Secondly, using such metrics on a grid system is in direct contrast to the platform independent nature of the grid execution model. This model very purposely abstracts the underlying platform to facilitate a ubiquitous execution environment. Furthermore, performance of the grid middleware affects the throughput of the system with little or bounded dependency on the execution platform. A thorough evaluation of the overhead imposed by the middleware is important and could be leveraged throughout the design cycle; by the developers and grid site administrators as a performance analysis and diagnosis tool and by grid users and grid application developers to find the optimal submission patterns for a particular grid site.

In this paper we present a black-box approach to performance analysis of grid middleware and introduce a non-invasive platform independent design of an evaluation tool that measures the overhead of the middleware and quantifies the effects of this overhead both on the throughput of the system and on the turnaround times of grid applications. The tool measures the overhead imposed by the grid middleware on different job submission patterns. It measures only the time spent on "grid work" for different number of simultaneous job submissions to show the effect of middleware overhead on the throughput of the system and the turn-around time of jobs for different submission patterns. Beside the useful information this tool provides to grid developers, administrators and users, we discuss how this approach could be leveraged to produce comparative performance analysis among different grid middlewares. With this approach we aim at producing middleware independent, comparable, reproducible and fair performance analysis of grid middlewares. As a proof of concept, we present the implementation details of the evaluation tool for UNICORE 5 and present the result of some initial experiments on the OMII-Europe [7] JRA4 internal evaluation test bed.

A discussion on how performance analysis of grid middleware and revealing the overhead imposed by middleware components provide indispensable information for grid developers, administrators and grid users is presented in section 2. Section 3 illustrates the design of our middleware evaluation tool. Section 4 provides implementation details of the tool for UNICORE 5. Section 5 demonstrates the result of initial experiments with the tool on UNICORE 5. Conclusions and future directions are presented in section 6.

2 Measuring the Overhead Imposed by the Middleware

To analyze the performance of the grid middleware rather than measuring the period of time an application spends on a resource, we propose a method to measure the time spent on the "grid work" per job, namely everything that has to be done before and after the job performs its computation on the grid resource. We expose the fixed costs of the grid middleware components-the latency of the components-and show how those costs affect the performance of a single job and ultimately the throughput of the entire system, i.e. the bandwidth. Although the overhead of grid middleware is not a constant-factor, it is bounded by the resource usage of the job (e.g. storage requirements); so useful patterns can be extracted from submitting different types of jobs with different resource usages.

Whether a large or a small job is submitted, the fixed costs contribute to the overhead of the grid system. From a user point of view, jobs that run for a long time amortize this cost versus the amount of computation that they perform. However, a frequent usage model of grids is to submit thousands of small jobs as part of a particular experiment (i.e. signal analysis, pattern recognition, Monte Carlo simulations). In order to facilitate better scheduling and resource utilization, users submit the jobs individually instead of all at once. However, this usage model is highly prone to the overheads induced by individual grid components and those components can contribute significantly to the run-time of a series of jobs representing a body of work that a user would like to perform.

Arguably, it is important to be able to record, diagnose and address this latency issue early in the design cycle. Grid developers can use this information for evaluating design choice trade-offs. Consequently, the analysis of the performance assessments that reflects the overhead imposed by the middleware could be beneficial to both developers to improve the design and implementation of the middleware components and to grid users to try to find the optimal usage model for the submission of their grid applications to a particular grid infrastructure. Given some estimates of the performance patterns of a standard installation of a middleware are published along with the middleware distributions, system administrators could also compare the result of performance measurements on their grid site and diagnose possible problems in their system configurations.

2.1 Comparison Across Different Middlewares

One of the things that contribute to the originality of our approach is the ambition to facilitate the production of performance data about grid components that is comparative among different middleware stacks. However, due to the differences among the middlewares there is no obvious way to produce this type of data with a high level of detail in a non-invasive manner. On the other hand, since such performance comparison has not been previously done, even somewhat crude measurements that can reveal differences in performance among the middlewares should be of interest to the Grid community.

A grid may from the point of view of this test be considered a black box, into which jobs are submitted and from which results are returned. Having this view, an initial thought is to put each of the middlewares in the place of the black box and record the submit time and the completion time, i.e. the time when the results are returned. Assuming that the middlewares are set up in a fair way (running on the same hardware, using the same number of worker nodes etc.) the results from such measurements provide a basis for comparison of different grid middlewares and also show which grid middleware handles a certain number of jobs in the shortest time-all in the scope of a particular set of use cases.

Obviously to provide a basis for sound and valid comparison, it is necessary to create reasonably fair conditions when performing comparisons between middlewares. However, fair conditions are very difficult, not to say impossible, to achieve since there are such large differences in how the middlewares are designed. We argue that adopting a best effort approach can still give us some comparable performance measurements across different grid middlewares.

3 Design of the Middleware Evaluation Tool

As an initial attempt to create an evaluation tool for grid middleware with the goals mentioned in the previous section, we propose a simple architecture for a non-invasive, middleware independent performance analysis tool that treats the middleware as a black box and gathers and visualizes performance data. This tool consists of scripts for submitting jobs to the middleware, recording the necessary timing information and for processing the output into graphs and charts. On the grid system, jobs are submitted, executed and the output is returned. Recording timestamps for when jobs are submitted, started, terminated and for when the output is returned, makes it possible to measure the overhead imposed on jobs by a middleware prior to and after execution (we call this pre-work and post-work). Information extracted from this measurement includes how much overhead the grid imposes on a submitted job and how this overhead changes when the number of submitted jobs varies.

We can assume that somewhere inside the grid, each job is executed and thus has an actual start and end time. Recording these times in addition to the submit and completion time, adds to the level of detail of the performance assessment. However, it is not obvious that these times are easily available, at least not considering the fact that the submit and completion times as well as the start and end times may be recorded on different computers on which clocks may differ significantly. In order to circumvent this problem, we use call-backs from the submitted jobs to the submitting computer. When jobs make call-backs at the time of starting and finishing, the times when these call-backs are received at the submitting computer are recorded. Obviously this approach introduces other possible sources of error, such as network latency, but initially it is assumed that the impact of these potential errors is negligible in a controlled environment. A schematic view of this execution case is illustrated in Fig 1. The performance measurements extracted via this approach gives a view of how different settings of a middleware performs and how different design choices affect the overall performance of the system and can also be further advanced to provide comparative performance data between different grid middlewares. It can be used as a starting point for where to instrument for more detailed measurements and could also help developers decide where to focus for improving performance.

For the measurements presented in this paper, the jobs that have been submitted are 'no-op' jobs, i.e. jobs with a negligible execution time. A natural continuation would be to extend this to include also other types of jobs, in order to stress particular components of the system. Further development of this approach would be to increase the number of jobs that are submitted and measure how the turnaround time changes. This can provide a view of how the middleware handles large workloads and also give a hint of where potential bottlenecks are to be found. The level of detail in this approach may not be high enough to point out exactly which components are the main performance bottlenecks, but gives an indication of where to spend effort on further investigation. These tools are being developed and tested on four different grid middlewares within the OMII-Europe Joint Research Activity on benchmarking [7]. As a proof of concept, the result of some initial experiment on UNICORE 5 is presented in section 5.

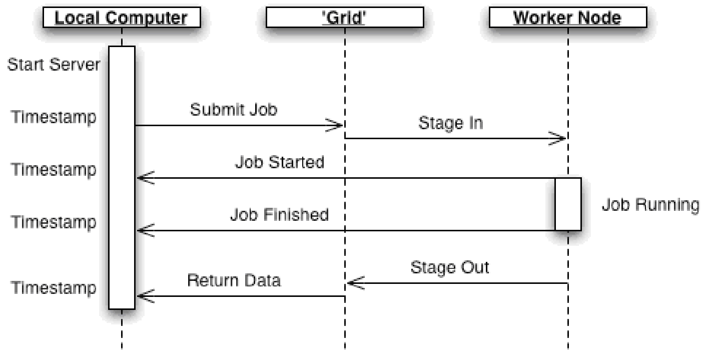


Fig. 1. Schematic view of the test script execution; jobs make call-backs when starting and finishing and the times when these call-backs are received at the submitting computer are recorded

3.1 Components of the Evaluation Tool

In this section, we describe the components of our evaluation tool:

Submit Script. A script to encapsulate the use of the scripting tools for the middleware and the recording of timestamps at relevant times. The submit script makes it possible for the user to submit a specified number of jobs to the middleware. The script produces the timestamps from the jobs as well as some metadata about the run. The output processor further processes this data to a more descriptive and graphic format.

Server. As described above, the submitted jobs issue call-backs to a server on the submitting computer when starting and stopping. The server listens for these call-backs and records timestamps when they are received. It is started by the submit script before submitting any jobs and when the last call-back has been received the server stops and hands over its output to the script.

Jobs. To perform initial measurement of the middleware overhead a very simple job without in-data or out-data is used. This job does not do anything else but issue the two call-backs to the server before it exits. This job can be easily extended to stress test or for more fine-grained performance analysis of the components.

Output Processor. A script to process the timestamps and produce different graphs to show the turnaround time, throughput and the pre-work overhead versus post work overhead imposed by the middleware.

4 Proof of Concept on UNICORE

As a proof of concept, we provide the implementation details of the test tool for UNICORE 5 and present the result of running some initial experiments with the tool on the OMII-Europe JRA4 internal evaluation infrastructure.

4.1 UNICORE

The UNICORE (Uniform Interface to Computer Resources) grid middleware is a software infrastructure that aims at providing the user with a smooth, secure, and intuitive access to their heterogeneous and distributed computing resources. The key objective of UNICORE is to enhance abstraction, security, site autonomy, ease of use and ease of installation. UNICORE, which was initially developed by the UNICORE and UNICORE Plus projects, is now one of the major projects in Grid computing [8,9,10]. Recently UNICORE 6 has been released in a beta version; it is a web service based implementation. In the near future we will extend our evaluation tool to perform tests also on UNICORE 6. This will produce comparative measurements between the two versions of the middleware.

4.2 Evaluation Testbed Setup

The UNICORE server has three main components: the Gateway, the Network Job Supervisor (NJS) and the Target System Interface (TSI). These may be placed on different nodes or on the same node in any combination. We have chosen to place each component on its own node and also put the client interface, "Command Line Interface Queue" (CLIQ), on yet another node. SSL is used for the communication between the client and the gateway as well as between the gateway and the NJS. The NJS and the TSI communicate over plain sockets. The TSI interfaces to a default installation of Torque.

4.3 Middleware Specific Details for the Test Tool

When submitting a large number of jobs, the proposed way to interact with UNICORE 5 is through CLIQ. It is a tool that handles both job submission, monitoring and output retrieval. It lets you specify jobs or entire workflows in an XML format. When copied to the CLIQ submission folder, the information in the XML job specification file is used to create an Abstract Job Object (AJO), which is then submitted to the UNICORE Gateway for execution. CLIQ keeps track of the status of the job and retrieves output files when the job is done.

The submit script interacts with CLIQ according to the following scheme:

1. Start CLIQ
2. For each job that should be submitted:
   (a) Create job description file.
   (b) Copy job description file to the CLIQ submit folder. This is where the submit timestamp is recorded.
3. Poll the folder where the output files from the jobs will be written. When the output files for a particular job is found, consider that job done. This is where the completion timestamp is recorded.
4. Stop CLIQ when all jobs are finished.

By default, CLIQ is restricted to not submit more than ten jobs at the same time and to poll for job status every fifth second. To stress the system slightly more we have set the maximum number of jobs to 1000 and to get more fine-grained results we have chosen a polling time of 100 ms. In next the section we present some results from initial tests as a proof of concept. We also categorize the results by interfacing UNICORE with different batch systems.

5 Experimental Results

The computer platform used consists of four nodes each with a 2.8 GHz Intel Pentium D CPU and 2 GB RAM connected through gigabit Ethernet. Scientific Linux is the operating system that has been used. The version chosen was 3.0.8. We used version 5 for UNICORE and for batch systems we used version 2.1.6 of Torque from its originator (we call it vanilla Torque) and the Torque version 2.1.6, which is packaged with gLite 3.0.2 (we call it gLite Torque).

5.1 Initial Test Scenarios

Below is the test scenarios for this paper:

– Measurements on Unicore-5 with direct submission (we call it Fork submission or unicore-nobatch),
– Measurements on Unicore-5 with submissions through Torque (gLite flavor Torque) and
– Measurements on Unicore-5 with submissions through Torque (vanilla).

For the scenarios above we produce the following graphs:

– Pre/Post work bar chart
– Turnaround time per job
– Throughput of the system (number of submitted jobs divided by the total time it takes for them to complete)

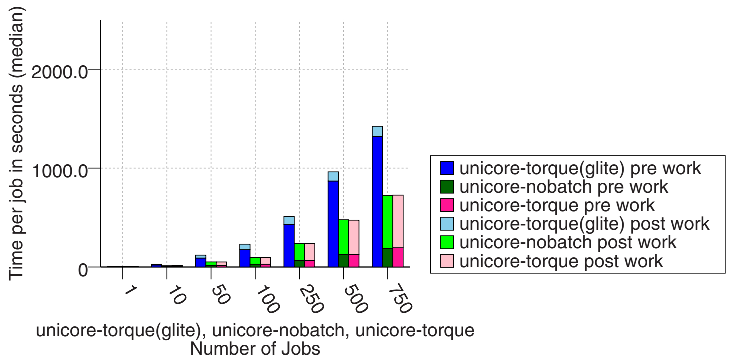


Fig. 2. UNICORE 5 Pre/Post work using Fork, Torque (vanilla) and Torque (gLite)

The figures in this section are generated using test data recorded by the testing tool. The data is then processed by a python script which then converts them into different graphs. From Fig 2. we see that the pre-work in case of Torque (vanilla) and no-batch is lower than the post work. For Torque (gLite) it shows that the pre work took more than half of the time of the post work. In Fig 3 we can notice that UNICORE performs equally when interfaced with Torque (vanilla) and with out any batch system (Fork)-measurements overlap in the figure. The line marked with triangles shows that UNICORE performs more slowly when interfaced with the gLite flavor of Torque. Fig 4 shows the throughput time and it is clearly shown the difference between UNICORE 5 when interfaced with Torque (vanilla) and when interfaced with Torque (gLite).

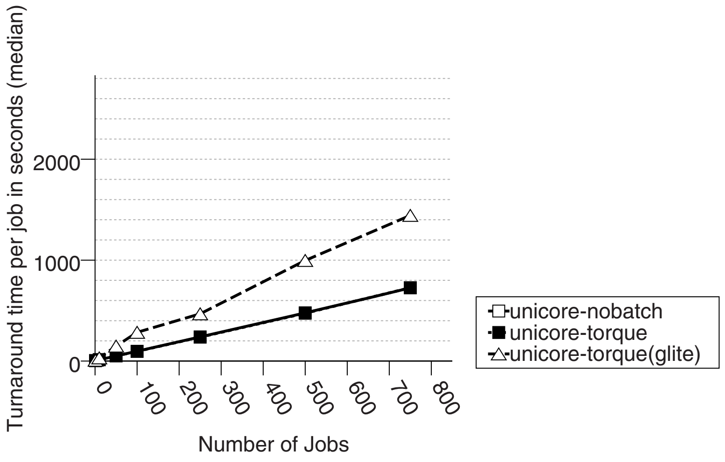


Fig. 3. UNICORE 5 turnaround time per job using Fork, Torque (vanilla), Torque (gLite). Note that lines from Unicore-torque and Unicore-nobatch are overlapping.

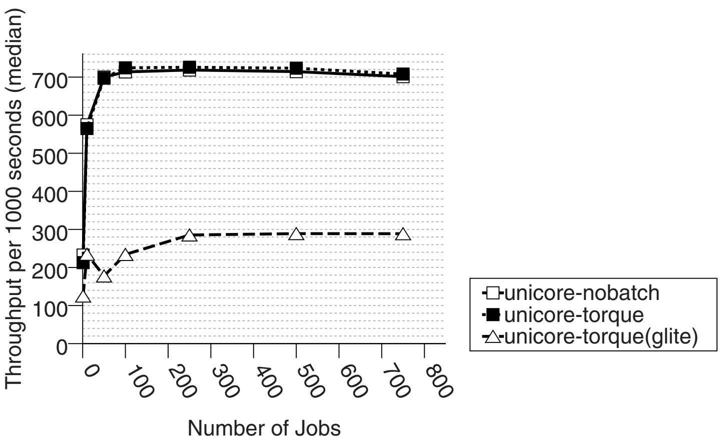


Fig. 4. UNICORE 5 throughput per 1000 seconds using Fork, Torque (vanilla), Torque (gLite). Note that lines from Unicore-torque and Unicore-nobatch are almost overlapping.

6 Summary and Future Work

In this paper we have presented a black-box approach to performance analysis of grid middleware and introduced a non-invasive platform independent architecture for evaluation tools to measure the overhead of the grid middleware and to quantify the effects of this overhead both on the throughput of the system and on the turnaround times of grid applications. Producing middleware component overhead measurements in a middleware independent and non-invasive manner and with high level of detail is currently difficult due to the differences among the middlewares. However, a use case that is relevant for all grids is the submission of jobs, their execution and the retrieval of their output data. Recording timestamps for when jobs are submitted, started, terminated and for when the output is returned, makes it possible to measure the overhead imposed on jobs by a middleware prior to and after execution and observe how this overhead changes when the number of submitted jobs varies.

As a future direction we intend to produce comparative performance data across different middlewares. Work is on-going for running tests using other middlewares including gLite, Globus and CROWN Grid in the context of the OMII-Europe Joint Research Activity on Benchmarking [7]. Extending the evaluation tool for UNICORE 6 and comparing the performance with that of UNICORE 5 is also planned.

We have provided a discussion on how exploiting this simple approach can produce performance analysis data that is beneficial to grid developers, users and grid site administrators. This approach can be further extended to provide comparative performance analysis across different grid middlewares. For proof of concept we presented the implementation details of the evaluation tool for

UNICORE 5 and the results obtained from running initial experiments with this tool. This approach is easily extended by introducing different types of jobs with specific resource requirements that would stress particular components of the middleware. To conclude, we have shown one approach to middleware independent and uniform performance evolution suites for grid middlewares that produce comparative performance measurements.

Acknowledgement

This work is supported by the European Commission through the OMII-Europe project-INFSO-RI-031844. For further information please refer to [7].

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Elsevier, Amsterdam (2004)
2. Nemeth, Z., Gombas, G., Balaton, Z.: Performance evaluation on grids: Directions, issues and open problems. In: 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2004), p. 290 (2004)
3. Snavely, A., Chun, G., Casanova, H., der Wijngaart, R.F.V., Frumkin, M.A.: Benchmarks for grid computing: a review of ongoing efforts and future directions. SIGMETRICS Perform. Eval. Rev. 30(4), 27–32 (2003)
4. Dikaiakos, M.: Grid benchmarking: Vision, challenges, and current status. Concurrency and Computation: Practice and Experience 19(1), 89–105 (2007)
5. Nmeth, Z.: Grid performance, grid benchmarks, grid metrics. In: Proceedings of the 3rd Cracow Grid Workshop, Cracow, pp. 34–41 (October 2003)
6. Tsouloupas, G., Dikaiakos, M.: Ranking and performance exploration of grid infrastructures: An interactive approach. In: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, Barcelona, September 28-29, 2006, pp. 313–315. IEEE Computer Society, Los Alamitos (2006)
7. Open Middleware Infrastructure Institute for Europe. Project no: RI03, –OMII-Europe (1844), `http://omii-europe.org`
8. The UNICORE project, `http://www.fz-juelich.de/zam/grid/unicore`
9. The UNICORE Plus project, `http://www.fz-juelich.de/unicoreplus`
10. UNICORE, Official website, `http://www.unicore.eu`

UNICORE/w3*

R. Menday and B. Hagemeier

Central Institute for Applied Mathematics,
Forschungszentrum Jülich, D-52425 Jülich, Germany
r.menday@fz-juelich.de

Abstract. Drawing on the core values of UNICORE - "seamless, intuitive and secure access" - in this paper we propose building the next generation of UNICORE by closely aligning the Grid and the Web, namely by using the Web as the homogenizing middleware layer for the Grid. The RESTful use of HTTP coupled with a unified, RDF-based model, results in a loosely-coupled, global scale architecture. Therefore the Grid as a very rich source of information contributes to the Semantic Web of data. This is the foundation for the strong focus on usability in UNICORE/w3. Navigation of the linked resources, filtering, powerful searching functionality, the annotation and sharing of resources and monitoring using Web syndication techniques are some of the features proposed.

1 Introduction

Key characteristics of the Web are that it is simple to understand and use, deeply integrated in the user's desktop and working practices, navigatable through linking, loosely coupled and has unparalleled support in terms of tools and support. The initial Web can be likened to a global document repository, the future evolution is towards the Web is a single global database. Human readability will be augmented with data published in a form that can be usefully processed by machines.

On the evolutionary path towards the Semantic Web [13], so-called Web 2.0 added a new dimension in the richness and interactivity of the experience for the user. It added customised content, and the notion that other users are 'out there' consuming the same content. Importantly, we have also seen progress on the machine readable Web characterised by Web sites offering 'web apis' explicitly designed for direct consumption by machines. This has enabled another feature found in Web 2.0 applications - the 'mash-up'. This is the combining and re-use of data published on the Web, to build new applications. This is often based on XML and JSON, so whilst good results can be achieved, e.g. Google Maps mash-ups, the free joining of data from sources distributed across the Internet is not fully realised. This can only really happen when a unified model for describing the data in the Web is in place. With the maturing of the Semantic Web we

* This work is partially funded through the European A-WARE project under grant FP6-2005-IST-034545.

are starting to see this happening. A result of this can be seen in the W3C Tabulator[11], an RDF browser allowing the user to explore the linked 'Web of data'. Furthermore, LinkingOpenData[6] is an interesting initiative targeting the 'bootstrapping' of the Semantic Web with the goal of making "... various open data sources available on the Web as RDF and to set RDF links between data items from different data sources". In future we will see more re-use, re-publishing, merging and integration of information over the Web. This is the Web as the global-scope, homogeneous layer providing a role similar to that of middleware for machine consumers of the data. Furthermore, as the Web evolves towards this Web of data, so to do the opportunities for using the Web as a middleware for Grid computing.

Why the Grid and the World Wide Web ?

Grid Computing is the "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations" [20]. It is fair to say that the initial target audience was the scientific community. It is interesting to compare with the initial impetus driving Tim Berners-Lee's vision of the World Wide Web. The Web was initially conceived due to a frustration preventing scientists from effectively collaborating and sharing scientific results[16]. Between the Web and the Grid there seems to be some shared purpose, although it is surprising that none of the main Grid middleware platforms have attempted to integrate, to a deeper extent, Grid middleware with the infrastructure of the Web.

Our reservations regarding the suitability of Web services for building a single Global grid, leads us to this paper where we present our vision for a Web-based Grid, as a starting point for discussion and future activity. We believe that building UNICORE as a Web application addresses the shortcomings of the previous versions of UNICORE, and provides new opportunities for our users.

Thus this paper makes the case for the single Grid - one which is realised as part of the Web. The vision is that every supercomputer (or any other computing resource) has a projection of it and its resources published and accessible over the Web. HTTP agents interact with these resources. The concept of navigating links (through clicking) to discover related information is highly intuitive for any computer user. With a resource-oriented approach we directly identify with a HTTP URI all our resources, for example, a File, Job, Storage, Reservation, User, etc. Given such a situation information on an executing job or a particular data artifact, for example, can be shared between collaborators or bookmarked. This is the participatory Grid where scientists are able to use the Grid infrastructure for the fine-grained sharing of data and jobs.

We cover many topics in the paper, and due to space restriction many technical details have had to be omitted. The paper continues as follows. In Section 2 we review some background to this work, in particular looking back at relevant architectural approaches and philosophy in the evolution of the UNICORE grid system since 1997. Section 3 covers some of the technologies which make the Web ideal for Grid computing. Section 4 describes the functionality and architecture of UNICORE/w3 - first by describing how the Grid of resources become part of the Web, and then by describing some possibilities for higher-level services which provide aggregation points exploiting these distributed resources. We review relevant work in Section 5, and offer a summary and outlook to the future in Section 6.

2 Review

The original goal of UNICORE was to develop production quality software for accessing the distributed resources of high performance computing centres, and largely this still drives UNICORE development today[1]. The reader is referred to [18] and [25] for good overviews. UNICORE/w3 draws on experience with all versions of UNICORE. Relevant architectural concepts are summarised below.

Abstract Job Object. The central architectural concept in pre-Web Services UNICORE (versions 1-5) is the Abstract Job Object (AJO). The AJO is the unified model of the distributed computing world. In this model computer resource descriptions and requests use the same model. It is realised as Java objects and uses the Java serialisation format as wire format[2]. UNICORE recognised early on that the Grid can be best modelled as a graph, and the AJO reflects this. The goal of seamless computing is addressed by the abstract-ness of the AJO. Incarnation in the AJO processing chain maps the abstractly expressed user request into an executable form for the targeted system.

The document-centric design of the AJO is almost exclusively procedural. Whilst this is convenient for modelling Jobs running on a batch system, it is interesting to note that the AJO does not contain a object representing a File on the Grid (an XFile is used to structure the responses to ListDirectory and FileCheck, but not as a first class entity of the model) making it difficult to capture at a top-level abstraction the notion that a Storage contains a set of Files[3]. This is an example of one of the shortcomings of the AJO that is addressed in current modelling.

Another consequence of the procedural model is the unconstrained nature of the 'operations' (or processes) encoded into the AJO. Changing operation names or subtle differences in how these documents are conveyed, means that a server upgrade almost always prompted corresponding client updates. This is satisfactorily in a tightly-controlled environment, but not in a Global setting where for example multiple versions of the same software should be expected to be in use. Thus, whilst UNICORE achieved vertical integration with the AJO, this was at the expense of loose-coupling.

[1] UNICORE 6 was released mid-2007 and is based on Web services technologies.

[2] A recognised issue with the implementation of the AJO model is the strong dependency on Java, and as a result of this the model is rather closed and difficult to extend. However, this is more of a simple rendering issue not fault with the model as such.

[3] We note that the AJO Portfolio object captures a subset of this requirement

Web-Services and XML. UNICORE 6 builds on top of a Web services based foundation developed in the UniGrids[2] project. Most other popular Grid middleware, for example Globus 4[3], has also migrated to a Web services based infrastructure in recent years. UNICORE 6 boasts excellent performance enhancements over previous UNICORE versions, and uses XML to overcome the Java serialisation format limitation. However, Web services were supposed to address the issues of tight-coupling in Grid architectures, bringing Grid infrastructure to the Internet scale. We claim that this has failed.

Much has been made of WSRF providing support for stateful Web services. Whilst the name of the Web Service Resource Framework (WSRF) seems to indicate a resource-centric approach, the resource in question is obscured by the service managing each WS-Resource. One re-occurring criticism of WSRF is that it is a distributed object system (perhaps more realistically a distributed object facade), and in essence it is. The state is hidden behind the service interface. Manipulation of the state of the WS-Resource is done through service-specific set of operations (although WSRF attempts to standardise some common state retrieval operations), and there is no explicit concept of navigatable state. In a Web services based SOA whilst there is a good loose-coupling of implementation (primarily through the use of XML) such that unlike the Java based AJO, in theory client and server do not have to be written in the same language. However, the strictly-defined and specific nature of the interface results in an early binding and consequent tight-coupling between client and server. A Service-oriented approach also makes the service the primary entity, and not the resources it manages. This has major implications regarding caching.

Whilst an improvement on the binary format of serialised Java objects, one can question semantic sparseness of XML as a document format. What emerges are XML 'islands' of information which make powerful search, query and joining of data difficult.

Therefore whilst the AJO does not make enough of its natural ability to describe a graph, in principal it is in a better position to model the richness and complexity of the Grid. We note that the graph-oriented modelling of the Grid evolved further in the UniGrids project [12], [23] and we take these ideas further in UNICORE/w3.

3 The World Wide Web

The standards driving the Web are absolutely established and stable. The same can be said for the Web infrastructure - HTTP servers, routers, firewalls, etc. This section describes some key characteristics and features of a Web oriented architecture.

Single Resource, Multiple Representations. As described in [17], we make the distinction between two kinds of resources: Information resources and non-information resources. Non-information resources are 'real world objects' which exist outside of the Web, but are identified by HTTP URIs. Therefore, rather than being directly de-referencable, following the advice of [24] we can use the Web as a lookup mechanism to find more information about these resources. We use the '303 URIs' method where a HTTP 303 redirects a HTTP request for a non-information resource, using content negotiation to redirect to the relevant representation of the information resource (for example, the RDF document describing the resource, or a human readable HTML representation). UNICORE/w3 should support HTML, XML, JSON and RDF representations as a minimal set.

REST and ATOM. Millions of Web sites are in a constant state of evolution but this happens without forcing the upgrade of each users' Web browser. Client and server are freer to evolve independently. The uniform interface constraint of REST[19] limits the set of allowable operations on resources, and REST advances this characteristic as one reason for the loose coupling of the Web. For HTTP, the essential operations are GET, PUT, POST and DELETE. As an example, taking the `getResourceDescription` and `getJobs` AJOs and migrating to one possible RESTful scheme, these become just GET on a VSite and VSite Jobs collection resources respectively. REST proposes a resource-oriented approach, where the state of a users interaction is reflected by the HTTP URIs they are currently 'talking to'. The state of an application is published as a navigatable set of resources, and through the navigation of links the state of the application is transferred back to the user as a URI. The agent can de-reference this URI to discover a new set of navigatable URIs. Furthermore, a resource-oriented approach is actually something very useful and beneficial from a user's perspective too as it is likely to achieve a desirable quality of 'bookmark-ability', and in the Grid domain, resources which a user might want to bookmark are very easy to identify. Furthermore, ATOM and the ATOM Publishing Protocol[1] offer a convenient payload for distributing Web content as a chronological sequence of items and for creating and updating Web resources. Both are milestone technologies and play an important role in this work.

The Semantic Web. By itself a Web based system based on HTTP and its URIs, with HTML, XML, JSON and ATOM representations is a significant advancement for UNICORE. Going further, the Grid is part of the Web, and the Web is a graph. An ability to describe this graph is extremely powerful as it brings the data together into a single unified model in a manner far more flexible than XML or Database structures. RDF is a language for representing information about resources, in the form of statements about these resources, including the type of a particular resource and the nature of the relationship with other resources. For example, in a supercomputer, a File is related to a particular Storage where it is held, and the nature of this relationship can be explicitly stated in a computer processable manner. Ontology (expressed using RDFS/OWL) is used to define the vocabulary for the resources, their types and the relationships between them. This can be used to infer non-explicitly stated relationships between the resources in the system, and the Semantic Web query language SPARQL[10] can be used to query this information.

4 Grid Computing with UNICORE/w3

UNICORE is the “Uniform Interface to Computing Resources” and it summarises its character very well. UNICORE/w3 software acts a gateway to the business and scientific resources of a single site, projecting representations of each resource onto the Web for those authorised to view and manipulate them. HTTP agents interacting at this interface perform the functionalities traditionally associated with Grid computing, namely resource management, information services and data management. All of which are dependent on a common security infrastructure. We explore each briefly in the following sections. Then we take a look at what is often referred to as ‘higher-level services’ - the Google and Yahoo Pipes of UNICORE/w3.

Information. Information publishing at a single resource (VSite in UNICORE terminology) is the publishing of information about the Grid resources at this particular site - Jobs, Storages, Files, etc. We use a lightweight ontology (inspired to some extent by the UniGrids ontology [12]) expressed using OWL to define the vocabulary for this description. We note that an attractive feature of RDF is that it encourages a data-first approach, whereby the collection of RDF statements can use vocabulary from a number of ontologies, and one is not constrained by defining a schema first and populating according only to that particular schema. In this sense the information model for UNICORE/w3 is highly flexible and extensible.

Resource Management. In the simplest case, resource management at a single VSite is achieved by POSTing activity descriptions to a Job collection resource using the ‘process this’ semantic associated with HTTP POST, resulting in the creation of new resources. The execution is managed with the various stages in the execution modelled as subordinate resources. An ATOM feed publishes the changes to the Job resource as it evolves. Complete Job removal uses HTTP DELETE on the Job resource which also deletes the subordinate resources. A possible enhancement is through publishing Application resources reflecting applications discovered on the target system (application software resources in UNICORE terminology). Application resources publish template request documents, bootstrapping resource negotiation.

Data Management. UNICORE/w3 enables a projection onto the Web of files and directories in storage systems. For UNICORE/w3 each entity is individually identified with its own HTTP URI. By default files in a storage are only accessible by the owner of the file. Basic information (modified times, size) and data management functionality (copy, move, delete) should be offered. Furthermore, it should be possible to instruct one site to initiate a transfer of a file to another remote storage. Following a mechanism similar to UNIX file permissions, we can allow the owner of a single file resource to make it accessible to a group of authenticated users. Furthermore, a file could be published as freely readable by everyone. This is similar to the publishing of Google Docs[4], which by default are private, but on request can be made viewable by all. Finally, the actual contents of a file should of course be retrievable. HTTP transfer is supported, but also a user may wish to take advantage of other transfer mechanisms, and XMPP and Bittorrent are potential candidates here.

Security. HTTP requests for a particular resource are subject to authorisation checking based on the URI of the resource and the identity of the authenticator. Controlling who is able to view/modify resources is important. For example, usage information from a group of users should only be accessible by administrators. Organising around resources makes an implementation of a robust and fine-grained authorisation policy cleaner. Authentication in such architecture also needs to be addressed robustly. Due to space constraints we have not analysed in depth the issues, but at the global scale, there is a growing need for a Web-native identity solution. We foresee leveraging the open, rapidly emerging OpenID standard. With OpenID, users identify themselves with a URI, aligning nicely with the RESTful, resource-centric design.

Higher-Level Functionality

For many Web users a search engine, such as Google, provides the 'jump-in' point to the Web. We see specialised search engines - indexing a subset of the Web related to a particular user's known Grid resources - offering a similar functionality for the Grid of resources of UNICORE/w3. Leveraging the rich graph of information and the search capability, UNICORE/w3 users will be able to view many presentations of their Grid of resources in ways most useful to them. These aggregation services, whilst accessing information from the underlying resources, can also assert additional RDF statements regarding these resources. Of course, such annotation carry less provenance, but can be used in many interesting ways. For example, the search engine can allow the user to tag particular resources in the underlying Grid of resources.

Searching and Tagging. Information from the Grid, once published as RDF, can be queried and joined with any information, including information from other resources. SPARQL[10] is the standard RDF query language. Provided with an RDF description of resources, SPARQL can be used to express queries across them. This may be used for a multitude of purposes - for resource selection, querying for support of a particular application, mining for trends, etc. As described above, the user is able to 'tag' particular resources with a free-form string. Queries can then take the tagging attributes on a particular resource also into consideration. Examples of typical queries include,

- find all Sites, current status and other useful monitoring information
- find the total number of Jobs executed at each site over the past 30 days
- find all Jobs tagged 'hot', which have appended files in the last 30 minutes
- find Files in directory with a specified file name pattern.

Once a new SPARQL query has been POSTed a new resource is created for that particular query. This can be re-run at any point by GETting that resource.

The results of a SPARQL SELECT query results in a table of information, each column header for each variable in the query and the results of the query in each row. Moreover an ATOM feed is associated with each query. Over time, changes in the results of the query are published in the feed allowing changes in the SPARQL result to be identified. We believe this to be a unique and extremely useful merging of SPARQL and Web syndication.

Workflow. Other services may offer to manage long-running executions, such as the coordinated usage of multiple computational resources. Following the REST principle that 'hypermedia is the engine of all application state' we have the foundation of a interactive and modifiable/dynamic workflow model. Workflow structures as RDF graphs are stored as additional statements related to the underlying resources. One can view a Web-based workflow as a collection of Web resources, and the state of the users interaction is captured by these resources, where every stage in the workflow is a resource in the graph, and can be de-referenced. This is a very powerful concept for when it comes to monitoring, tracking and re-writing.

5 Related Work

In this paper we propose using the Web and HTTP as a end-to-end infrastructure for the Grid, and we therefore note differences between this and many projects building portals over existing Grid infrastructures, for example, GridSphere[5]. The Semantic Grid[7] has made many explorations in the area of Semantic technologies applied to Grids. Much work from this working group is connected with adding Semantic support to Web services infrastructure to build Semantic Web Services, although recent activity at the OGF[8] is exploring synergies between Web 2.0 and Grid including a workshop at OGF 21, where we see our work as highly relevant. Of course a number of others have done some extremely interesting work in the application of Semantic and Semantic Web technologies to Grids. In 2001, [21] evaluated the applicability of using RDF schema as a vocabulary for Grid resources, and in [26] the authors propose using Semantic Web technologies for resource selection. In [14] a delegation ontology is derived and SPARQL is used to evaluate policy.

A proposal for layering SPARQL functionality and a HTTP interface over existing Grids appeared in [22], and this can be seen as forerunner to this work. Regarding other work in the direct UNICORE community we are not aware of any related work which builds upon the core UNICORE values, wholly recommending building on a RESTful Web foundation and using the Semantic Web technologies to add rich structure.

6 Outlook and Summary

In this paper we have outlined our vision for a Web-based future for UNICORE. We can summarise a number of possibilities for further exploration. The use of

OpenID[9] as an identity solution looks very appealing, but needs further analysis. For example, provision for delegated authorisation could possibly leverage the additional functionality offered by OpenID 2.0. For the searching functionality provided by the information services we would like to investigate other RDF query languages (of which there are many). We would also like to iterate further on the core UNICORE/w3 ontology, re-using existing ontologies where appropriate. A negotiation process to guarantee a particular quality of service (QoS) is usually managed using WS-Agreement[15] in SOA based Grids. We note that the Web naturally provides much of the capabilities necessary for resource negotiation - the notion of the Web as protocol state machine, and a template repository and would also like to examine this area in more detail. Finally, the nuts-and-bolts of taking the UNICORE/w3 concepts and making a production system from the current prototype adds a number of other topics for further discussion.

From the notion of clicking to navigate, to receiving a HTTP URI in an email, the Web is an extremely familiar environment for most people. Using the Web the middleware layer for the Grid is provided by HTTP. The constrained interface of HTTP, together with a RESTful design of the resources, offers much in terms of genuine loose-coupling between client and server, cacheability, scalability, stability and accessibility. In addition using the Semantic Web provides a basis the for flexible and powerful higher-level services going well beyond todays state of the art. We offer searching and the notion of the participatory Grid where scientists are empowered to easily share results computed on the Grid.

It is worth noting that an early prototype already demonstrates a number of these concepts. It is the strong conviction of the authors that the Web (more precisely the Semantic Web) will subsume the Grid as we know it today. Building on the core values of UNICORE - "seamless, secure, and intuitive access" - this paper is intended as a sincere exploration of the further opportunities which will come with a full migration to the Web.

References

1. ATOM, `http://tools.ietf.org/html/rfc4287`
2. European UniGrids Project, `http://www.unigrids.org`
3. Globus, `http://www.globus.org`
4. Google, `http://www.google.com`
5. GridSphere, `http://www.gridsphere.org/`
6. Linking Open Data, `http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpen Data`
7. OGF Semantic Grid, `http://www.semanticgrid.org/`
8. Open Grid Forum, `http://www.ogf.org/`
9. OpenID, `http://openid.net/`
10. Sparql, `http://www.w3.org/TR/rdf-sparql-query/`
11. The Tabulator, `http://www.w3.org/2005/ajar/tab`
12. UniGrids Ontology, `http://www.unigrids.org/ontology.html`
13. W3C Semantic Web, `http://www.w3.org/2001/sw/`

14. Ahsant, M., Basney, J., Mulmo, O., Lee, A., Johnsson, L.: Toward An On-demand Restricted Delegation Mechanism for Grids. In: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, September 28th-29th, Barcelona (2006)
15. Andrieux, A.: et al. Web Services Agreement Specification (2007)
16. Berners-Lee, T.: Weaving the Web (Hardcover). Texere Publishing Ltd. (November 1999)
17. Bizer, C., Cyganiak, R., Heath, T.: How to Publish Linked Data on the Web, `http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/`
18. Erwin, D. (ed.): UNICORE Plus Final Report – Uniform Interface to Computing Resources. UNICORE Forum e.V. (2003), ISBN 3-00-011592-7
19. Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD thesis, Chair-Richard N. Taylor (2000)
20. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. In: Sakellariou, R., Keane, J.A., Gurd, J.R., Freeman, L. (eds.) Euro-Par 2001. LNCS, vol. 2150, Springer, Heidelberg (2001)
21. Gunter, D., Jackson, K.: The applicability of rdf-schema as a syntax for describing grid resource metadata (2001)
22. Menday, R., Streit, A.: SPAQRLing UNICORE. In: Proc. of the 6rd Cracow Grid Workshop (CGW 2006) (2006)
23. Parkin, M., van den Burghe, S., Corcho, O., Snelling, D., Brooke, J.: The Knowledge of the Grid: A Grid Ontology. In: Proc. of the 6rd Cracow Grid Workshop (CGW 2006), October 15–18 (2006)
24. Sauermann, L., Cyganiak, R., Vlkel, M.: Cool uris for the semantic web. DFKI Technical Memo TM-07-01 (2007)
25. Streit, A., Erwin, D., Lippert, T., Mallmann, D., Menday, R., Rambadt, M., Riedel, M., Romberg, M., Schuller, B., Wieder, P.: Unicore - From Project Results to Production Grids. In: Grandinetti, L. (ed.) Grid Computing and New Frontiers of High Performance Processing, Elsevier, Amsterdam (2005)
26. Tangmunarunkit, H., Decker, S., Kesselman, C.: Ontology-based resource matching in the grid—the grid meets the semantic web (2003)

Chemomentum - UNICORE 6 Based Infrastructure for Complex Applications in Science and Technology

Bernd Schuller[1], Bastian Demuth[1], Hartmut Mix[2], Katharina Rasch[2], Mathilde Romberg[3], Sulev Sild[4], Uko Maran[4], Piotr Bała[5], Enrico del Grosso[6], Mosé Casalegno[7], Nadège Piclin[8], Marco Pintore[8], Wibke Sudholt[9], and Kim K. Baldridge[9]

[1] Research Centre Jülich, Jülich, Germany
[2] Dresden University of Technology, Dresden, Germany
[3] University of Ulster, Coleraine, Northern Ireland
[4] University of Tartu, Tartu, Estonia
[5] ICM, Warsaw, Poland
[6] TXT e-Solutions, Milan, Italy
[7] Istituto Mario Negri, Milan, Italy
[8] BioChemics Consulting, Orléans, France
[9] University of Zurich, Zurich, Switzerland

Abstract. Chemomentum, Grid Services based Environment to enable Innovative Research, is an end-user focused approach to exploit Grid computing for diverse application domains. Building on top of UNICORE 6, we are designing and implementing a flexible, user-friendly Grid system focussing on high-performance processing of complex application workflows and management of data, metadata and knowledge. This paper outlines Chemomentum vision, application scenarios, technical challenges, software architecture and design of the system.

1 Introduction

The European Chemomentum [1] project[1] joins today partners from nine institutions, developing an environment for workflow-oriented scientific as well as industrial applications. A wide range of end-users in science and technology with varying IT and Grid computing expertise is targeted. The Chemomentum system is designed to be generic and thus usable in a wide range of application domains. In the applications we are targeting, data management is of crucial importance. Secure access to stored data, metadata-based lookup, global identifiers and location management comprise a few of the many challenges in this area of research and development.

Within the project, a chemical application domain in the natural and life sciences and material science, is targeted. Domain-specific additions are provided

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

that will enable the building of complex workflows for predictive modelling tasks in risk assessment, biotechnology, material and drug design. Our aim is to test-drive the system together with the installed applications in the context of the European REACH legislation [2], offering *in silico* tools that may prove crucial in reducing costs and the number of animal tests. The Chemomentum extends the experience of partners gained in the OpenMolGRID [3,4], EUROGRID [5], Gemstone [6], and DEMETRA [7] projects.

At the forefront of our efforts is high usability. Therefore, we follow a user-centric approach that allows for seamless and interactive usage of software by domain experts. A flexible client architecture, based on the Eclipse platform, will enable us and the domain experts to deliver focused end-user interfaces that combine ease of use with detailed access to the system's features. For special use cases, web based access will be provided. Furthermore, a client library will allow the Chemomentum system to be coupled to any type of domain-specific user interface.

To realise such a system, we selected UNICORE 6 [8] as Grid middleware, providing basic Grid services such as security, job execution and file transfer. Furthermore, it allows to write and deploy additional services, such as a high-level workflow processing system and data management services.

The remainder of the paper is organised as follows: In section 2, we describe the characteristics of the different application domains and scenarios that the Chemomentum software is targeted towards, from which requirements can be extracted. Section 3 explains in detail the architecture of the Chemomentum system. Section 4 provides an overview of the actual testbed infrastructure that can be used to test-drive the Chemomentum framework. Finally, section 5, concludes the publication and discusses the status and plans.

2 Application Scenarios

The main application drivers for Chemomentum originate from the chemical and biochemical application domains, in particular drug and material design, predictive modelling, and toxicological and environmental risk assessment of chemicals. In this application domain, a wide array of different technologies is used for computer based modelling. As a second, complementary application scenario supply chain planning will be used to verify the generality of the Chemomentum approach.

The Chemomentum project aims to implement generic interfaces for the following application families from biochemistry and chemistry:

– Linear and non-linear predictive model building;
– 3D coordinate generation for molecules from their connectivity information;
– Chemical conformational space analysis;
– Semi-empirical and *ab initio* quantum chemical calculations;
– Molecular descriptor calculation;
– Prediction of chemical property and activity values;

– Data filtering and preprocessing: clustering and artificial intelligence for rational chemical data pre-treatment;
– Molecular docking;
– Homology modelling of proteins.

A significant challenge in Chemomentum is the integration of such diverse application families in a way that enables the user to be creative and explorative, so that innovative ways of using existing tools for a variety of application scenarios are possible. This is achieved through the Chemomentum workflow engine (see section 3.1). In the following, details about the use of these applications in certain application scenarios are described.

Quantitative structure-activity relationship. (QSAR) methods are key technologies behind computer based modelling of chemicals, including chemical and physical properties and biological activities. The QSAR methodology assumes that the activity of chemicals is determined by their molecular structures and that there exists a mathematical relationship, $P = f(s)$, between them, where P is the modelled activity and s is the numerical representation (i.e. molecular descriptors) of the molecular structure. Molecular descriptors in this context are used to calibrate various chemical properties or biological activities among a grouping of chemicals. QSAR is a complex application scenario. The process starts with the design of data sets for the model development and predictions which may require access to a variety of heterogeneous data sources. This is followed by the geometry optimisation of molecular structures (see paragraph on Quantum Chemistry) and the calculation of molecular descriptors. Both of these steps are data parallel and therefore ideal candidates for Grid computing. Once the experimental values and molecular descriptors are available, statistical methods are applied for the development of the actual mathematical model. Finally the developed QSAR models can be used for predicting activity values of chemicals from their molecular descriptor values. Our previous examples of the use of Grid technology in QSAR are the modelling of aqueous solubility [9], HIV-1 protease inhibitors [10], and acute toxicity [11].

Quantum chemistry. (QC) represents a collection of theoretical and computational methods that employ approximate solutions to the Schroedinger equation $\hat{H}\Psi = E\Psi$ to characterise the structure, properties and mechanism of reaction of molecular systems. The utilisation of QC approaches is of particular importance for addressing questions involving chemical reactions where bonds are broken or formed, and to obtain results with very high accuracy. Prominent application areas in both academia and industry include the elucidation of reaction mechanisms, the prediction of molecule properties (e.g., as part of QSAR procedures), and the computation of, for example, the effects of solvation or spectroscopic information. Unfortunately, depending on the level of their theoretical foundation, QC methods can be quite complex and computationally intensive. This means that such calculations are typically not carried out without significant intervention of the user, and in many cases require knowledge in the field in order to properly set up and carry through computations to address questions of structure and reactivity in molecules. QC approaches are also often combined with approaches from classical mechanics, to enable the treatment of larger (e.g., biological or material) systems. Together with high-throughput investigations involving many molecules, these are the main application areas where QC can benefit from Grid computing, data, and workflow processing (see, e.g., Refs. [12,13,14] for earlier approaches). A major challenge for Grid computing in QC is that many of the highly used computer programs are legacy, and that each has their own input and output formats. However, there are now European efforts under way to agree on standardised data-exchange formats [15]. The quantum chemical program packages that will initially be tackled within the Chemomentum project include MOPAC [16] and GAMESS [17].

Molecular docking. describes the procedure of finding optimal structural fits of one component molecule within another, typically a ligand in a protein pocket. Most importantly, docking is used in the early stages of the pharmaceutical drug design pipeline, and has thus gained considerable interest from academia as well as industry. Typically one begins with a previously determined 3D structure of a protein with a certain biological function, and investigates possibilities for modification of that function. This could involve a functional modification of a ligand already embedded in the protein, or, the 'docking' of a small molecule (i.e., a potential drug candidate). The idea is to use computation to search for an optimal orientation of the two molecules (i.e., usually a binding of the ligand at the natural activity site of the protein). In this procedure, each calculation step involves computation of the free energy of the complex formation, to be determined by an empirical function. Such calculations are usually embarrassingly parallel, and therefore highly suited for high throughput-oriented Grid computing. A clear demonstration of this is the Docking@Home [18] project. Often, a number of preparation and analysis steps is necessary, for example, to apply more accurate energy functions to score the ligands, after an initial set of complexes has been identified. As such, molecular docking also profits from workflow approaches. A typical representative for a widely-distributed docking program package is AutoDock [19]. However, molecular docking methods are still topic of intensive research, in particular to obtain better scoring functions. At the University of Zurich for instance, we are currently developing a Grid-enabled docking procedure based on quantum chemistry and biomolecular continuum electrostatics [20].

Homology modelling. is a fundamental and widely used tool in molecular biology. Using such tools, biologists are able to compare DNA or protein sequences from the same or different organism. In this way, evolutionary relationships between organisms can be explored, and biological functions of new sequences can be predicted. The main task is to find statistically significant local similarities between pairs: a user-defined (protein or DNA) sequence and sequences from databases.

All of these types of application scenarios are used in many aspects of chemistry and life science related fields. Prominent examples include:

The European REACH policy. (Registration and evaluation of chemicals). Substances have to be carefully evaluated and assessed for possible risks to human health and the environment prior to marketing and distribution. This requires massive efforts in terms of time, money and animal testing. The Chemomentum system has the potential to reduce the need for animal testing as such by providing a wide variety of solutions for the computational, *in silico*, testing of chemicals. QSAR has been selected as one of the key predictive modelling techniques for REACH.

Research and development in the pharmaceutics. Computational solutions are more and more used to virtually screen large databases of compounds in order to identify potential leads concerning a given biological activity. The size of the databases requires extensive computing solutions that can be achieved via efficient use of Grid technologies. In particular, emphasis should be given to the high level of security while working with the data with significant intellectual property risk.

Research and development in biotechnology. The biotechnology sector is using the largest databases for screening and determining technologically relevant new substances. This determines their need for the extensive computational solutions to reduce the time to product and also the costs.

The scope of Chemomentum system is not limited to scientific applications. The overall architecture is generic and will be also tested on completely unrelated applications in the retail sector, such as demand prediction in supply chain management.

Supply chain planning. The purpose of this application is to apply the capabilities offered by Chemomentum to a complex scenario where different and heterogeneous applications need to work together in a common environment. The scenario is based on the tools offered by TXT e-Solutions (TXTPERFORM suite [24]), but it has a sufficiently generic degree of validity. The focus involves three different phases: 1. Demand and merchandise forecasting; 2. Assortment and allocation plan; 3. Replenishment planning. These phases are usually connected together to form a complex workflow that involves the usage of different applications. This scenario is posing some major challenges: First, in the solution that is currently deployed, maintenance costs are high due to usage of different applications requiring heterogeneous environments. A Grid based integration solution can be expected to reduce costs by providing seamless and unified access to these heterogeneous enviromnents. Second, the control of the entire workflow needs to be human driven, which is the only possible way to have full control on the behaviour of the applications.

The solution that Chemomentum is investigating is based on workflow automation, where applications are located behind an infrastructure that allows a common access to their functionalities and resources.

3 Architecture of the Chemomentum System

This section provides a birds-eye view on the Chemomentum architecture, and the primary associated functional systems. These functional systems will be explored in more detail in the subsequent sections.

As Figure 1 shows, Chemomentum consists of five major sub-systems.

- Workflow System: Executing and managing workflows, for details see section 3.1;
- Data Management: Services for accessing data and metadata, for details see section 3.2
- Grid Management: Monitoring and managing all services making up the Chemomentum systems [2];
- UNICORE 6 Hosting Environment: Hosts fabric services (such as job execution) provided by the UNICORE Grid middleware. The additional Chemomentum services will make use of the UNICORE 6 hosting environment, to benefit from its features and integrate with the UNICORE 6 security, see section 3.3;
- Client Layer: graphical interfaces for end-users and administrators for accessing and managing Chemomentum services, for details see section 3.4.

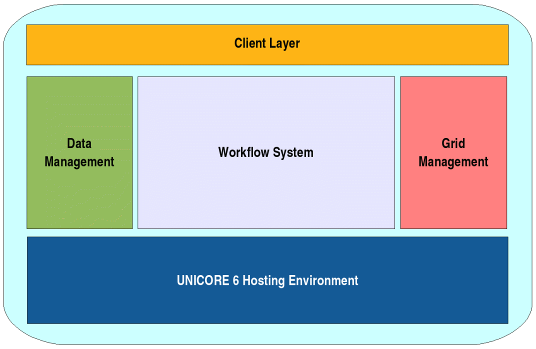


Fig. 1. Overview of the Chemomentum Architecture

3.1 Workflow Processing

Workflow processing is at the core of the Chemomentum system. As Figure 2 shows, the Workflow System is subdivided into two layers of abstraction: the Workflow Engine and Service Orchestration layers. The Workflow Engine processes a workflow on a logical level, whereas the Service Orchestrator deals with the actual execution and supervision of tasks using different services and resources on the Grid. Thus, the workflow processing logic is cleanly separated from the re-occurring invocations of low level Grid services.

[2] The Grid management components are not covered in detail in the present paper due to space constraints.

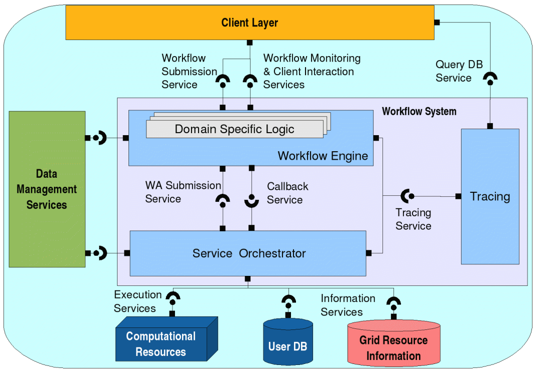


Fig. 2. Three-tier approach to workflow

In order to handle different domain specific workflow languages, the workflow engine translates incoming workflow descriptions to a generic workflow language using pluggable Domain Specific Logic (DSL) modules. Splitting and distribution of computational effort in order to maximise throughput and to make most efficient use of the available computational resources is one of the core goals of the Chemomentum system. Each activity of the translated workflow results in an atomic unit of work for the Grid, a so called "Work Assignment". Work Assignments (WAs) are abstract, in the sense that they are not bound to specific service endpoints on the Grid. They are individually submitted to the Service Orchestrator for execution. Due to dependencies and conditions in the translated workflow, WAs cannot be executed in arbitrary order. For instance, one WA may depend on the output data of another WA. The Workflow Engine keeps track of such preconditions and does not submit WAs with unmet preconditions. The Service Orchestrator transforms each incoming WA into a job, given in Job Submission Description Language (JSDL). It exchanges the logical names of input files for addresses of physical file locations. It submits the job to a computing resource, supervises job execution and informs the workflow engine of job completion or failure.

3.2 Data Management

The objectives of the Data Management System (DMS) within Chemomentum are to provide data storage and retrieval functionality and to give a global data view independent of actual data location. The DMS establishes a distributed data store that can be used to house data and, even more important, corresponding metadata that thoroughly describe the data stored. The metadata include typical descriptive information like for example the user who produced the data, the date of generation/modification or the applications that were used to produce the data. Additionally, meta information specific to a domain can be stored, e.g. the list of properties used in building a QSAR model. The set of domain specific metadata items to be stored is flexible.

A key feature of the Data Management System is the user-transparent access to external data sources of different kinds, e.g. web-based databases, flat files, etc. The data from the internal data store and the external databases is presented to the requester in data views specifically adapted to its demands. This could be, for example, a view that focuses on a specific domain or application and integrates the information gathered from external databases with data stored in the internal data store.

The DMS is accessed by the Workflow System when executing a workflow. Input data necessary for executing the workflow is retrieved, output data generated by the workflow and the metadata that describes this output data is stored. The end-user can access the data management system to browse the data in the internal data store – highly aided by the extensive metadata; access data in external databases, or manually upload interesting files to the internal data store.

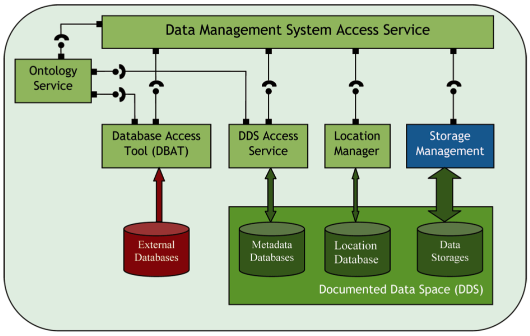


Fig. 3. Data Management System architecture

The heart of the Data Management System is the Documented Data Space (DDS), see Figure 3. It is composed of Metadata Databases, Data Storages and a Location Database. The Data Storages contain data in flat files, typically input and output data produced by Chemomentum. The Location Database acts as a global file location directory by indexing those files and assigning them globally unique logical names. The Metadata Databases contain metadata that describe the files in the data storages, referencing them by their logical names.

The central interface to the DMS is the Data Management System Access Service. It forwards service requests to the appropriate service(s), collects the results and returns them to the requester. For example, in case of a data store request containing files and metadata it
(a) instructs the Storage Management Service to stage out the files from the temporary data store to one of the Data Storages in the DDS;
(b) instructs the Location Manager Service to index the uploaded files in the Location Database and generate logical names; and
(c) instructs the DDS Access Service to insert the metadata into one of the Metadata Databases in the DDS.

The Database Access Tool (DBAT) serves as a uniform interface to external databases. It transforms a query to the data management system into the native query language used by the external database, queries the external database and transforms the result back into a format the client understands. The Ontology Service supports the DBAT in providing information, e.g. synonyms of molecular names, to broaden queries to external data sources. It provides knowledge about types and vocabulary necessary to interpret data retrieved from external sources or to store domain-specific data. The metadata vocabulary is not fixed. Given that different scientific fields and simulation environment have different needs, new domains can easily be added to the system to meet these needs and existing domains can be extended. For each domain a vocabulary is maintained, which does not only include metadata items with their respective data types, but also further properties like a human readable name or information on its range. Currently the domain vocabularies are provided as a relational database schema. Metadata and knowledge representation models like RDF[25] and Topic Maps[26] that could allow an even more expressive description of the metadata of a domain are under investigation for use in Chemomentum.

Users of the data management system do not have to know about the vocabulary of a domain beforehand. Existing domains and their vocabularies can be queried for and retrieved whenever needed. This allows to develop generic clients, e.g. a data browsing client that enables the user to browse through stored metadata regardless of the domain.

3.3 Security Considerations

The present system adopts the security infrastructure of UNICORE 6, which is based on X.509 certificates, and offers fine-grained access control to services based on user attributes stored in a user database (XUUDB). This is sufficient for simple use of UNICORE 6, for example to decide whether a given user has the permission to run a job or access a storage service.

However, Chemomentum has more complex requirements on security. For example, trust delegation mechanisms are obviously needed for Chemomentum to support the secure use of the chain of services needed to process a workflow. The Explicit Trust Delegation model of UNICORE 5 [27] is a first step, but becomes impractical for larger Grids. At the time of writing (June 2007), the final trust delegation model in UNICORE 6 was not fully defined.

Also, the Data Management System needs access to even more detailed user permission data, similar to the level of detail offered by a relational database system, in order to provide fine enough access control. Thus, the access control available in UNICORE 6 is not sufficient.

Similarly, the user management in UNICORE 6 as available in June 2007 is not sufficient for large Grids, and complex virtual organisation structures. Thus, Chemomentum aims to provide a security service offering Virtual Organisations (VO) based user management and storing extended user attributes for making access control decisions. This is intended as a drop-in replacement for the basic UNICORE 6 solution.

Irrespective of the security middleware in place, it is well known that in certain application domains data can be so sensitive that the data owners will not allow to send it over a network. Chemomentum caters for these users by allowing private data to be stored and maintained physically at a specific site, in a private instance of the DDS service discussed in 3.2.

3.4 Clients

The client layer of the Chemomentum system provides the associated interface to end-users in the different targeted application domains (see section 2) and to administrators of the infrastructure. In this way, the Chemomentum services can be easily and uniformly accessed and managed. Our main focus is on graphical user interfaces, but for certain user communities, command line interfaces might be of interest. We are developing three types of clients: (i) Eclipse [28] Rich Client Platform-based client, (ii) web portal and (iii)third-party clients.

As a common application programming interface between the clients and the server-side infrastructure, a Basic Chemomentum Client Library (BCCL) has been created. The BCCL will, in particular, enable third-party tools, for example domain-specific user interfaces such as Gemstone [6], to integrate into the Chemomentum system.

As an initial step towards building the Chemomentum graphical user interface, GridBeans [21], based directly on the underlying UNICORE environment, have been developed and already deployed in the testbed (see also the next section). GridBeans can later be included into the rich client, or used independently. A generic GridBean suitable for any type of UNICORE-deployed application has been developed and successfully tested with the quantum chemistry package GAMESS [17]. Furthermore, a GridBean for the BLAST application is available [22]. It is designed to provide a user interface organised in a similar way as the one on the NCBI-BLAST [23] website. The main purpose is to provide ease of use, particularly by scientists who are used to the NCBI website and may not want to learn about a different interface. Also, GridBeans for QSAR model development have been developed.

4 The Chemomentum Testbed

To evaluate and test the software, Chemomentum provides a special site [29], which allows the wide public to test new system capabilities. Everybody can use the test installation to evaluate the software. The site allows to request a test certificate, valid for the test site, and when certificate is issued it allows do run grid jobs on a dedicated virtual system. The testbed is intended to run quick jobs.

Setting up of a pilot installation available to a wide user community involves a security challenge. In order to minimise the risk we have used a virtualisation technique to protect the server integrity and to offer a reasonable security level. The execution system has been set up on an isolated virtual host, and be treated as compromised all the time. A further benefit of the virtualisation technique is the possibility to share the images of the virtual systems, which allows for fast and easy replication of the pilot installation.

UNICORE provides client applications to access distributed resources. These clients offer a powerful and extensible interface to the resources, but a number of users expect web access to the Grid. This motivation, clearly expressed in the Chemomentum project, resulted in the redesign of the UNICORE Client framework to allow easy integration with portals. A client library has been created which allows for fast development of web interfaces to the Grid resources.

5 Conclusions and Outlook

Driven by application requirements, we have designed a UNICORE 6 based Grid system focussing on ease of use, high-performance processing of complex scientific and industrial workflows and elaborate data management solutions.

We have found UNICORE 6 to be an excellent basis for such a system, as its open structure makes it easy to build custom services and integrate them into the basic infrastructure. The security requirements of Chemomentum were not fully met by the beta version of UNICORE 6, but we expect marked improvements in this area with the availability of the final UNICORE 6 release.

The presented design has been validated in a number of prototype implementations. A first implementation of the full workflow processing stack is available, based on the Shark [30] XPDL workflow engine. The Data Management System in the first prototype provides storage and retrieval of data from the QSAR domain with predefined metadata. A client component allows to browse through data stored in the DDS.

The first release of the basic software framework, based on UNICORE 6 final, is targeted for August 2007, which will be available on the public testbed in October 2007.

References

1. Chemomentum: Grid-Services based Environment for enabling Innovative Research, `http://www.chemomentum.org`
2. REGULATION (EC) No 1907/2006 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 18 December 2006 concerning the Registration, Evaluation, Authorisation and Restriction of Chemicals (REACH), establishing a European Chemicals Agency, amending Directive 1999/45/EC
3. OpenMolGRID: Website, `http://www.openmolgrid.org`
4. Sild, S., Maran, U., Lomaka, A., Karelson, M.: Open Computing Grid for Molecular Science and Engineering. J. Chem. Inf. Model 46, 953–959 (2006)
5. EUROGRID: Website, `www.eurogrid.org`
6. Gemstone: Website, `http://gemstone.mozdev.org`
7. Benfenati, E. (ed.): Quantitative Structure-Activity Relationships (QSAR) for Pesticide Regulatory Purposes. Elsevier, Amsterdam (2007)
8. UNICORE: Website, `http://www.unicore.eu`

9. Sild, S., Maran, U., Romberg, M., Schuller, B., Benfenati, E.: OpenMolGRID: Using Automated Workflows in GRID Computing Environment. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 464–473. Springer, Heidelberg (2005)
10. Maran, U., Sild, S., Kahn, I., Takkis, K.: Mining of the Chemical Information in GRID Environment. Future Gen. Comput. Syst. 23, 76–83 (2007)
11. Maran, U., Sild, S., Mazzatorta, P., Casalegno, M., Benfenati, E., Romberg, M.: Grid Computing for the Estimation of Toxicity: Acute Toxicity on Fathead Minnow (Pimephales promelas). In: Dubitzky, W., Schuster, A., Sloot, P.M.A., Schröder, M., Romberg, M. (eds.) GCCB 2006. LNCS (LNBI), vol. 4360, pp. 60–74. Springer, Heidelberg (2007)
12. Sudholt, W., Baldridge, K.K., Abramson, D., Enticott, C., Garic, S., Kondric, C., Nguyen, D.: Application of Grid Computing to Parameter Sweeps and Optimizations in Molecular Modeling. Future Gen. Comput. Syst. 21, 27–35 (2005)
13. Baldridge, K.K., Greenberg, J.P., Sudholt, W., Mock, S., Altintas, I., Amoreira, C., Potier, Y., Birnbaum, A., Bhatia, K., Taufer, M.: The Computational Chemistry Prototyping Environment. Proceedings of the IEEE 93, 510–521 (2005)
14. Sudholt, W., Altintas, I., Baldridge, K.: Scientific Workflow Infrastructure for Computational Chemistry on the Grid. In: Alexandrov, V.N., et al. (eds.) ICCS 2006. LNCS, vol. 3993, pp. 69–76. Springer, Heidelberg (2006)
15. COST Action D37 Grid Computing in Chemistry: GRIDCHEM, Working Group 0004 DeciQ
16. MOPAC: Website, `http://www.openmopac.net`
17. GAMESS: Website, `www.msg.ameslab.gov/GAMESS/GAMESS.html`
18. Docking@Home: Website, `docking.utep.edu`
19. AutoDock: Website, `autodock.scripps.edu`
20. Seebeck, F.P., Guainazzi, A., Amoreira, C., Baldridge, K.K., Hilvert, D.: Stereoselectivity and Expanded Substrate Scope of an Engineered PLP-dependent Aldolase. Angew. Chem. Int. Ed. 45, 6824–6826 (2006)
21. GPE4GTK Project: Website, `http://gpe4gtk.sourceforge.net`
22. Borcz, M., Kluszczyński, R., Bała, P.: BLAST Application on the GPE/UnicoreGS Grid. In: Lehner, W., et al. (eds.) Euro-Par Workshops 2006. LNCS, vol. 4375, pp. 244–252. Springer, Heidelberg (2007)
23. National Center for Biotechnology Information (NCBI), BLAST interface, `http://www.ncbi.nlm.nih.gov/BLAST/`
24. TXT e-solutions website, `http://www.txt.it`
25. Resource Description Framework, `http://www.w3.org/RDF/`
26. Steve Pepper, The TAO of Topic Maps, `www.ontopia.net/topicmaps/materials/tao.html`
27. Snelling, D.F., van den Berghe, S., Qian Li, V.: Explicit Trust Delegation: Security for Dynamic Grids. FUJITSU Sci. Tech. J. 40(2), 282–294 (2004)
28. Eclipse: Website, `http://www.eclipse.org`
29. Chemomentum Portal, Demo site, `http://www.chemomentum.org/chemomentum/demo/`
30. Shark Workflow Engine, `shark.enhydra.org`

Flexible Streaming Infrastructure for UNICORE

Krzysztof Benedyczak[1], Aleksander Nowiński[2], 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 Modelling
Warsaw University
Pawińskiego 5a, 02-106 Warsaw, Poland

Abstract. We present recent innovation in a field of advanced, multipurpose streaming solutions for the grid. The described solution is based on the Unigrids Streaming Framework [7] which has been adopted to the UNICORE 6 middleware and extended. The main focus of this paper is the UGSF Data Flow Editor, which is an universal tool for powerful streaming composition. It has been developed to provide users with a graphical interface for streaming applications on the grid.

1 Introduction

Data streaming is one of the most advanced services available in the grid. The data streaming, as well as instrument and application steering, gets significant attention of the grid users and middleware developers. Unfortunately, there is still lack of good and stable solutions ready for wide deployment.

The early works on the data streaming in the UNICORE [1] were focused on solutions dedicated to the particular applications. Among others, specialized approaches based on the SSH and pre-OGSA version of UNICORE have been developed [2]. Currently SSH tunnels are used in COVS framework [3] to perfom streaming of visualisation data. However those developments can not be seen as universal streaming framework, as are limited to concrete applications.

The new version of OGSA [4] and therefore service oriented version of the UNICORE opened the possibility to develop much better solutions. As a result the UniGrids Streaming Framework (UGSF) has been made available. The UGSF is a middleware, which serves as an engine for the new generation of UNICORE streaming services. It contains also a library for the client development.

The UGSF established a new quality in comparison to the solutions developed for pre-web services versions of UNICORE. The large part of the UGSF - the *UGSF core* is not directly operated by the end-users. It contains a number of stream implementations which can be deployed and used without any additional effort. However, from the user's point of view, there were still few problems in the UGSF. The main one is lack of user-friendly, graphical tools to access the streaming services. The development of such tools was limited either to the programming from scratch[1] or to use GPE GridBeans technology [5]. The first solution is obviously hard, while the second one could be efficiently used only for concrete streaming scenarios.

The shortcomings of the existing solution motivated us to start the development of the UGSF Data Flow Client. It can be considered as an alternative to the UGSF clients, which have been developed in the GridBeans technology. The UGSF Data Flow Client can be seen as a streaming counterpart of the GPE Expert Client, which allows for composing arbitrary *live* stream connections instead of building jobs workflows.

In this paper we present the UGSF platform and describe the Data Flow Client. The detailed discussion of the plugin interface is performed. Example applications are presented in the second part. The final part of the paper describes ongoing and future developments.

2 The UGSF Platform

The aim of the UniGrids Streaming Framework (UGSF) is the provision of a direct data streaming for applications. The main part of UGSF is *UGSF core*, which is a middleware that allows developers to create dedicated streaming services. Every system based on the UGSF will use the *core* together with some application dependent code. The UGSF core provides basic functionality common for all streaming applications. This includes a creation or a shut down of a connection. UGSF contains also a large group of versatile software pieces which can be reused when creating actual implementations of streaming services. A good example is a component which allows for locating UNICORE job's working directory.

The detailed UGSF architecture is presented elsewhere [6] and here we will introduce its brief overview focused on the basic elements. The UGSF is built based on the fundamental concept of *stream*. A Stream is a logical entity that encapsulates some functionality on server-side. This functionality can be:

1. production of data (by any means, e.g. importing it, reading from a file or even creating it)
2. consumption of data (any kind of destination like file export to another streaming system)
3. data processing (filtering, changing data format, etc.)

The stream defines at least one of input ports, output ports or bidirectional ports. Those represent possibility to connect to different UGSF streams or ordinary clients. The stream can have many ports, with different characteristics (list of allowed formats is a good example of stream characteristics). It can also perform many logical data transfers - “streamings” in common sense. Every logical

[1] “Programming from scratch” refers here to the GUI part of such applications. The UGSF provides client-side library, which simplifies development of the logic of the application. Nevertheless, significant programming skills are necessary.

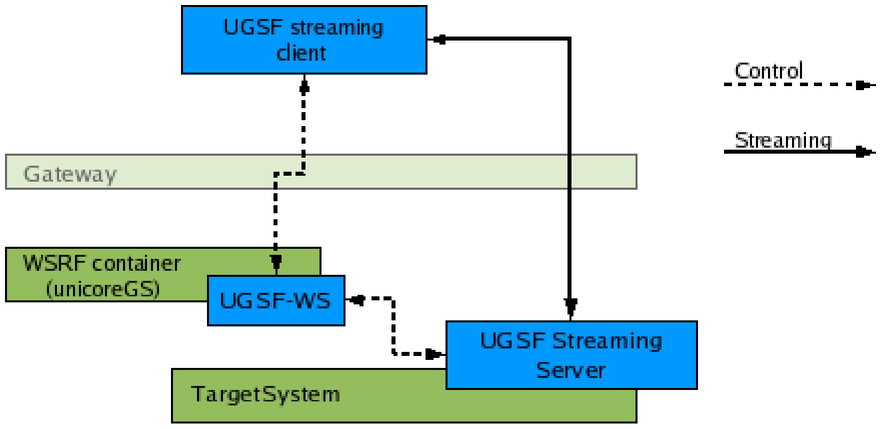


Fig. 1. The general architecture of UGSF

data transfer, started or ended at port of the stream is called a *flow*. To give an example: "Theora video decompressing stream" can have two flows; one input which accepts compressed Theora video, and one output flow, that transfers raw video frames. In this example the stream has defined data processing while neither production nor consumption of data (as defined above) are present. The implementation that provides defined functionality is called a *stream implementation*. Usage of the stream implementation requires creation of a corresponding resource which is called a *stream instance*.

The UGSF streams (or better *streams implementations*) have a metadata attached. For every flow there are defined capabilities such as reconnect capability, supported formats (or protocols) and others. It is possible to specify more than one format for a single flow as well as to express the only supported formats' combinations for the all flows of the stream together.

2.1 UGSF Architecture

The UGSF system is based on the WSRF compliant version of the UNICORE, (version 6) [1]. The first version of UGSF system was developed using UnicoreGS software [7] as the hosting environment. Recently, the hosting environment was changed to use mainstream UNICORE 6 server-side components: WSRFLite and UnicoreX.

The *UGSF core* consists of a *UGSF Web Service* part, *Streaming Server* part and a library to create clients. The usage of the last component is optional. The *UGSF Web Service* takes advantage of WSRF capabilities. It is used to control a set of available stream types, to create new streams and to manage already created ones. The *Streaming Server* part is managed by a *UGSF Web Service* and performs streaming. Client library is used to simplify the creation of the client-side software. Overall architecture is shown in Figure 1.

The *UGSF core* is complemented with stream implementations. Those consists of two server-side parts: streaming server and web service modules. Web service module implements control operations specific to the stream implementation. Streaming server module deals with a wire streaming protocol and data consumption or acquisition. The recent works provided a possibility to add client-side implementation for the stream implementation. The details are given in the section 3.

2.2 UGSF Web Service

The *UGSF Web Service* component consists of two kinds of web services. A base one (called *StreamingFrameworkService*) is responsible for connection authorization, creation of stream and its setup. During this process the new WS-Resource (called *StreamManagementService*) is created by a dedicated web service interface. This WS-Resource acts as a controller of an active streaming connection.

The *StreamingFrameworkService* is a WS-Resource which maintains list of *StreamManagementServices*. The *StreamingFrameworkService* allows users to get a list of available stream instances and to set up a connection to the specified one. The list of both owned and accessible streams is available. In addition, the *StreamingFrameworkService* has an administrative interface, which empowers system administrator to enable and disable particular stream types on the fly. The service reconfiguration such as addition or removal of stream types is also possible.

For each created stream an instance of the *StreamManagementService* allows user to perform universal operations for all streams. This includes shutting the stream down (by means of WS-Lifetime interface) or getting status and statistics of the connection. This functionality can be easily enriched by the developer. He can extend *StreamManagementService* with additional operations. The enriched implementations are free to consume any special XML configuration supplied to the *StreamingFrameworkService* and required for service setup and creation.

2.3 UGSF Streaming Server

The UGSF *Streaming Server* is a stand-alone, modular application which performs streaming to and from the target system. The server is tightly connected with the *UGSF Web Service* which maintains stream definitions (however *UGSF Web Service* can control multiple *Streaming Servers* without a problem). The server is modular and configurable.

Streaming Server modules can be divided into two categories: entry point modules and stream modules. The first kind of modules is responsible for implementation of special handshake protocol used to start streaming connection. Thanks to the modular architecture there can be many of such protocols available, even concurrently. An addition of a new one is possible and easy. Currently HTTP and HTTPS entry modules are available (special connection parameters are passed in HTTP header).

The second category of modules is responsible for streaming implementation. These modules can operate simultaneously in both directions: pushing the data from a server or pulling to the server. Stream module implements required elements of functionality presented in section 2. One possible class of streams are "filtering" streams which provides neither source nor sink for data. The data is read from a client, then processed and finally written out to a (possibly another) client.

Integration of the streaming functionality with grid jobs is of great interest here. The UGSF, among its standard stream modules, supplies visualization stream implementation. It can stream any kind of file both from and to job's USpace determined based on the given UNICORE job's reference and file's name. There is also a set of other implementations available, including TCP tunnels, UDP over TCP tunnels or multiplexer which clones input into many copies to name a few.

3 UGSF Data Flow Client

The UGSF Data Flow Client has been designed to provide solid base for creation of a specialized client applications which have to deal with *complex* streaming scenarios, including arbitrary data flow composition. The basic idea was to provide support for all, or nearly all generic features of UGSF and to support features available in sophisticated stream implementations by pluggable modules. Graphical approach was chosen for manipulating stream instances connections (i.e. data flow). To be fully functional, Data Flow Client must have possibility to act as a local endpoint for streaming in addition to control server to server connections. Local machine should be able to stream data in both directions to and from UGSF servers.

3.1 Generic Functionality

Data Flow Client manages the user's keystore which allows access to the grid. As the first entry point, some sort of resource discovery must be performed to locate streaming services. This is achieved in the usual manner for the UNICORE 6 — the user has to provide addresses of the registries. The same registry can be used for both UNICORE and UGSF services. The content of registries is automatically fetched and UGSF services are enumerated in side panel, called *services panel*. User can choose between having all services displayed or only those which are present in the actually chosen registry.

The stream instances managed by UGSF services are displayed in the same *services panel* in the form of a tree. From the context menus the user can create new instances, and destroy existing ones. While destroying is simple, the creation of a new stream instance is a more advanced operation and involves configuration of the instance. The client shows a pop-up dialog similar to shown in fig. 3. The dialog allows user to choose among all stream types defined in the selected streaming service. Further on, the user can choose the name for the job, set the

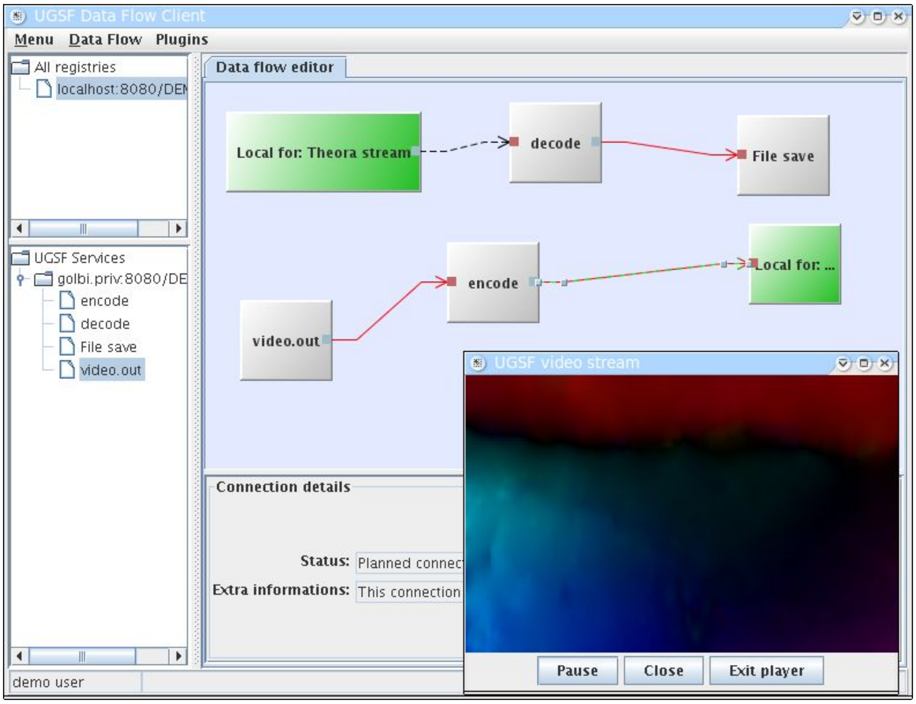


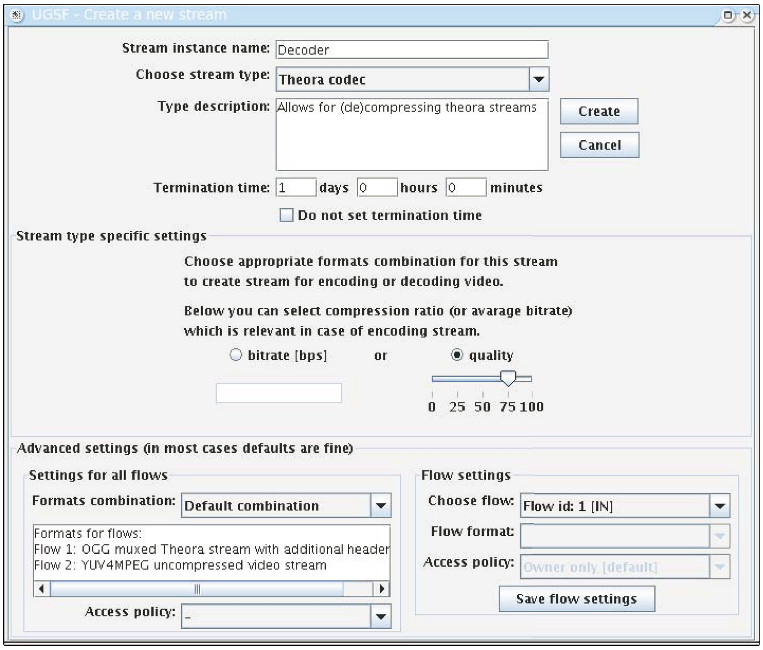
Fig. 2. The example of Data Flow Editor usage: Two simple data flows are prepared; one streams a local video file to a remote grid node, where UGSF filtering stream decompress the video and pushes it to the input file of the grid job. The job processes the input and its output is (after compression) sent back to the client, where live results are presented (optical flow of the input video sequence in this example).

termination time and select initial set of formats to be used. The dialog makes use of plugins to render GUI for preparation of any stream specific configuration that is needed.

When stream instance is created, its entry appears in the *services panel* tree. From there the stream instance can be added to the main workbench of the program: the graphical data flow editor. The editor visualizes data flow as a directed graph. Vertexes symbolize stream instances. Every vertex can have multiple *ports*, i.e. places of edges' attachment. Edges are used intuitively to represent stream connections (flows). Each fundamental property of a flow specifies direction of data transport.

Every stream can have many flows and adequate number of ports. There are also different kinds of ports: input, output and bidirectional rendered in a slightly different color at a different side of a vertex.

Usage of a stream instance in the data flow editor is accomplished by context menus. There are two menus: one for the stream instance to perform stream manipulation, and another one for every port to invoke operations related to the particular flows. For example, they allow for creating a new connection. The


UGSF - Create a new stream

Stream instance name: Decoder

Choose stream type: Theora codec

Type description: Allows for (de)compressing theora streams

Termination time: 1 days 0 hours 0 minutes

☐ Do not set termination time

Stream type specific settings

Choose appropriate formats combination for this stream
to create stream for encoding or decoding video.

Below you can select compression ratio (or avarage bitrate)
which is relevant in case of encoding stream.

○ bitrate [bps] or ● quality

0 25 50 75 100

Advanced settings (in most cases defaults are fine)

Settings for all flows

Formats combination: Default combination

Formats for flows:
Flow 1: OGG muxed Theora stream with additional header
Flow 2: YUV4MPEG uncompressed video stream

Access policy: _

Flow settings

Choose flow: Flow id: 1 [IN]

Flow format:

Access policy: Owner only [default]

Fig. 3. Dialog used to create a stream instance in Data Flow Client. The parameters of video compression are set in subpanel which is provided by a plugin. The rest of the dialog window is generic.

client does not allow to connect two ports (an therefore flows) with incompatible formats or data transport directions.

An automatic policy setting is used for created connections and on default only creator can connect to the stream. This leads to the problem whenever one stream instance created in Data Flow Client has to connect to another instance. In this case the Data Flow Client sets up permissions behind the scene. The client fetches the identity of the stream A which initiates connection. In the next step it changes the authorization policy of the target stream, to accept the connections with the A.

There is also a possibility to manually control the most of generic features of the stream from the vertex, port and edge context menus. For example, if the stream supports multiple formats, which is the generic UGSF feature, it is possible to change them. The streams in the UGSF allow for choosing a format of the individual flow or of all flows together. Whenever there are format dependencies, a change of a format of one flow will affect the format of another.

The UGSF Data Flow Client also provides features to monitor data flow state. This is the function of a dedicated panel located below the data flow editor panel. The information about selected element is displayed there. If a port is selected, then related flow properties are shown including (among others) flow status CONNECTED, DISCONNECTED, statistics of transferred bytes in any direction, average transfer speed and the time of last activity. The status is updated either automatically or manually by the user.

The Data Flow Client also provides auto-discovery of data flows. This is necessary to avoid problems, when there are stream instances created by other clients or in other sessions. The available streaming services allow for doing this except for the connections between UGSF streams and external clients. The auto-discovery process can be divided into two parts. The more simple one is used whenever stream instance is added to the data flow editor. The instance is checked if it has some active connections. If so, the another side of connection is determined. If it is already known by the editor, the connection is automatically added. If it is not present, an "orphaned" connection is drawn to mark that the stream is used.

There is also a more advanced feature which finds all vertices with orphaned edges, tries to locate them on the grid (not only among streams in the editor) the peers and add them to the editor. The process is repeated until no more orphaned connections exist or there is no known element to be added.

3.2 Plugins

The generic functionality of the UGSF cannot be used without dedicated stream implementations which offer specialized control and configuration possibilities. The Streaming Framework Client uses dynamically loaded plugins to manage stream implementations.

There are cases where plugin for the stream implementation is not needed. The example is multiplexer stream which does not need any special configuration because it uses only standard operations of UGSF platform.

The client offers extensibility points for the plugins. Usage of most of them is optional and then some default values/components are used instead. The UGSF Data Flow Client can provide:

– GUI to ask for these stream creation parameters which are implementation dependent. The GUI has to return those parameters. It is used in stream creation dialog, for example to specify compression parameters for Theora stream encoding (see fig. 3).
– Menu with operations related to the stream. Such a menu is added to the context menu of vertex in data flow editor panel.
– Menu with operations applicable to the particular flow of the stream. Such a menu is attached to an appropriate port menu of the stream.
– Ability to create local endpoints.

Local endpoint feature is designed to allow user's machine to act as a peer in a data flow. In the UGSF the description how to connect to the remote stream is implementation dependant. Implementation provides data to stream and describes data format or application protocol. Local source or sink of data needs to be defined as well. This functionality is reserved exclusively for the plugins.

When local endpoint is created, it appears in a data flow editor as a vertex (but of different color than ordinary stream instances). The functionality resembles the simplified stream vertex — there are ports and two kinds of context menus. The only difference is that the contents of the menus are coming nearly exclusively from plugin implementation.

To give an example, the IVis stream used to stream files to/from UNICORE job work directory allows for creating local endpoint. This endpoint offers two features activated in its context menu: to create output flow and to create input flow. In the first case local file needs to be specified for streaming it out. In the latter case, the name of a local file to store streamed data is required.

4 Related Work

In general there are few general frameworks which integrate computing grid infrastructure with advanced streaming capabilities and flexible data flow creation.

In the case of UNICORE platform there is no such solution known to the authors. However frameworks that offers (some) streaming capabilities exist. One example is COVS framework [3]. It's aim is to support online visualisation and steering of scientific applications run on the grid, with collaboration support. The COVS implementation uses VISIT library [8] as underlying technology. Therefore COVS application is available only for VISIT enabled software. The COVS framework uses SSH to tunnel VISIT protocol and extedns it with web service management capabilities. The data flow is fixed: one application run on the grid node can be steered and visualised by one or more end-users. The UGSF approach to the streaming is far more powerful. It does not restrict streaming to one (eg. SSH) low level protocol. Usage of other protocols can bring large performance gain, especially when encryption is not required. UGSF allows for client $\leftrightarrow$ server communication as COVS does, but also for server $\leftrightarrow$ server. Last but not least, UGSF can be used with any streaming application run on the grid, not only those VISIT enabled. In conclusion we can state that UGSF and VISIT overlap only in small part of functionality. UGSF provides low level mechanism and COVS could be built on top of it.

The length of this paper doesn't allow for performing through comparison with streaming frameworks for other than UNICORE grid platforms as NaradaBrokering [9], GridKit [10] or GATES [11]. However we can state here that, whilst most of such platforms offers very extensive features in case of streaming itself, their integration with computational grid is very limmited. Also the visual data flow editor described in this paper is a significant advantage of UGSF compared to other solutions.

5 Conclusions and Future Work

The presented solution is a big step forward in providing streaming capabilities for UNICORE. It is a convenient and easy base to be used for universal stream composition.

Some of the presented streaming features can be performed using standard UNICORE technology such as GridBeans and UNICORE Clients. This is not possible for special cases, where user wants to execute workflow and stream data between tasks executed on the different target systems. With the UGSF Data Flow client, such task can be built with a few mouse clicks.

The solution presented here solves the most important problems related to the data streaming in the grid but needs some further development. One of the important features is possibility to save and restore data flow composition, which is different from saving graphical representation of workflow as it can involve many complicated situations. One example is recreation of already destroyed stream instances and connections. This might require development of the dedicated service acting as data stream broker. The another required feature is UGSF administrative interface, which will allow to define stream types and deploy and manage streaming services. Such work is now in progress.

This work was partially supported by European Commission under IST grant UniGrids (No. 004279).

References

1. UNICORE project (May 2007), `http://sourceforge.net/projects/unicore`
2. Benedyczak, K., Nowiński, A., Nowiński, K., Bała, P.: Real-Time Visualisation in the Grid Using UNICORE Middleware. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 608–615. Springer, Heidelberg (2006)
3. Riedel, M. et al: Requirements and Design of a Collaborative Online Visualization and Steering Framework for Grid and e-Science infrastructures. German e-Science Conference, (May 2007)
4. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project, 2002 (May 2007), `http://www.globus.org/alliance/publications/papers/ogsa.pdf`
5. GPE4GTK project (May 2007), `http://gpe4gtk.sourceforge.net`
6. Benedyczak, K., Nowinski, A., Nowinski, K., Bala, P.: UniGrids Streaming Framework. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, Springer, Heidelberg (2007)
7. UniGrids project (May 2007), `http://www.unigrids.org`
8. Visualization Interface Toolkit (May 2007), `http://www.fz-juelich.de/zam/visit`
9. Pallickara, S., Fox, G.: NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, pp. 41–61. Springer, Heidelberg (2003)
10. Grace, P., et al.: GRIDKIT: Pluggable Overlay Networks for Grid Computing. In: Meersman, R., Tari, Z. (eds.) CoopIS/DOA/ODBASE (2). LNCS, vol. 3291, pp. 1463–1481. Springer, Heidelberg (2004)
11. Chen, L., Reddy, K., Agrawal, G.: GATES: A Grid-Based Middleware for Processing Distributed Data Streams. In: HPDC, IEEE Computer Society, Los Alamitos (2004)

Extending UNICORE 5 Authentication Model by Supporting Proxy Certificate Profile Extensions

Katerina Stamou[1], Fredrik Hedman[1], and Anthony Iliopoulos[2]

[1] Center for Parallel Computers (PDC), KTH, SE-100 44 STOCKHOLM, Sweden
{kstamou,hedman}@kth.se
[2] Department of Computer Science, School of Systems Engineering, The University of Reading, Whiteknights, READING RG6 6AY, Uk
ailiop@teilam.gr

Abstract. Authentication interoperability between the UNICORE grid middleware system and other Grid middleware systems is addressed. An approach to extending the UNICORE authentication model to support a proxy certificate (RFC3280) profile is presented. This optional feature can then be enabled based on site policy. Furthermore, the addition capacitates further advances related to authorization. With interoperability becoming a key issue in many production environments, extending the generality of UNICORE in this way opens up the possibility of direct and general interoperability scenarios.

Keywords: UNICORE, proxy certificates, RFC 3280, X.509, grid, interoperability, authentication, authorization.

1 Introduction

This paper describes an implementation of proxy certificate profile [1] authentication support in the gateway component of the UNICORE release 5 grid middleware system. The motivation behind this effort is to enable and promote functional interoperability among the different existing grid systems currently in production [2].

The existing authentication model of UNICORE allows for end-to-end plain X.509 mutual authentication between clients and UNICORE services. The proposed extension introduces support for verifying and authenticating against proxy certificates, thus allowing a wider and more flexible range of credential management in the UNICORE architecture. Numerous kinds of setups will benefit from the appearance of the features that the proxy certificate profile offers, the most significant being a controllable restricted form of trust delegation. This enables single sign-on like functionality allowing the possibility of disconnected operations that many of the other grid middleware systems already include. Environments that have a large grid deployment base and make use of UNICORE in parallel with other grid systems can benefit from the proposed functionality enhancement.

All work conducted pertains to UNICORE release 5, and is generally directed towards the pre-WebServices era systems.

2 The UNICORE Authentication Model

The UNICORE 5 release [3] is the well-tested, production-ready and pre-Web Services version of the UNICORE Grid middleware [4]

The system supports end-to-end user X.509 certificates, as main tokens of authentication. The UNICORE gateway service acts as the central entrance point for task submissions, and takes care of authenticating incoming client requests based on their presented X.509 certificates. The gateway maintains a configurable list of trusted Certificate Authoritiy (CA) certificates, thus clients are considered trusted and successfully authenticated only if their X.509 certificate is signed by a trusted CA.

Following a successful authentication, the gateway forwards the user X.509-signed Abstract Job Object (AJO) task to a configured Network Job Supervisor (NJS). There takes place the authorization part. The NJS detects if the X.509 certificate that signed the supplied AJO task exists in the Unicore USer Data Base (UUDB). All entries in the UUDB are comprised by an end-user X.509 certificate and a UNIX login account name. Upon successful authorization, the requested task executes having the privileges of the local UNIX user that the UUDB entry maps to.

Overall, this process is able to operate with ordinary X.509 certificates. There is no provision for restricted privilege delegation[1] which is present in other Grid middleware efforts (e.g. gLite, Globus, Naregi) and proves to be a highly desired functionality for Grid operations. This static credential management scheme is secure but lacks some flexibility. The proxy certificate profile approach comes to cover this functionality gap and offer a finer-granularity control.

3 Implementation of the Proposed Enhancement

Proxy X.509 certificate extensions provide a means to achieve trust delegation and restriction of delegated privileges [1]. In general terms it forms a simple and adequately secure approach that can be easily supported by extending an existing SSL/TLS-enabled server.

In our development setup, we have used SUN's JDK release 5 and OpenSSL toolkit, release 0.9.8d. For applications written in the C/C++ programming language the OpenSSL library [5] is the most widely used; it has full support for proxy certificates. The Java API does not natively support proxy X.509 extensions. However, the Java language provides a way to override the default TrustManager methods [6] which are responsible for making the authorization decisions during the relevant SSL handshake phase. Because third-party external

[1] Except from the "Explicit Trust Delegation" scheme, which does not offer the flexibility of proxy certificate extensions.

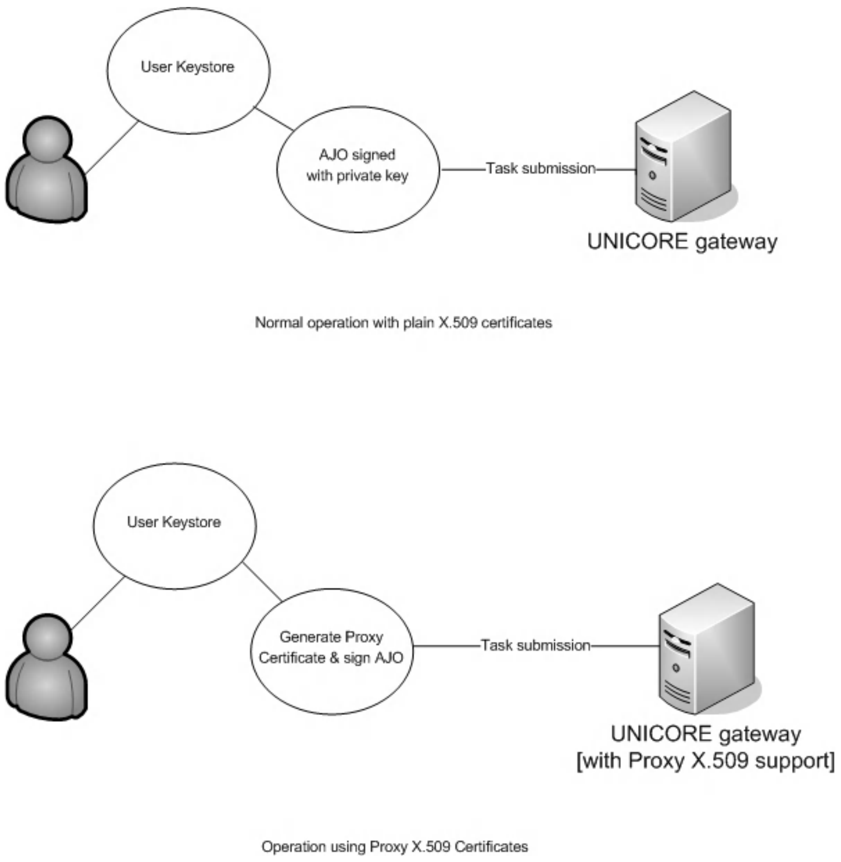

User Keystore
AJO signed with private key
Task submission
UNICORE gateway
Normal operation with plain X.509 certificates
User Keystore
Generate Proxy Certificate & sign AJO
Task submission
UNICORE gateway [with Proxy X.509 support]
Operation using Proxy X.509 Certificates

Fig. 1. Diagram of operation with & without proxy certificates. During normal operation the user's private key from the keystore is used to sign the AJO task to be submitted. In the proposed extension, a proxy certificate is generated and then used to sign the AJO.

libraries, such as BouncyCastle [7], have yet to include full support for proxy certificates, implementing a customized TrustManager class is thus a solution to access the functionality provided by proxy certificates from Java.

The relevant TrustManager method in the UNICORE gateway responsible for the authentication decisions is the `checkClientTrusted` method. Our customized version of this method works as follows:

– Initially, it instantiates a default TrustManager which attempts to verify the supplied client certificate chain, in order to cover authentication of connections using non-proxy X.509 certificates.
– If this fails, it attempts to verify the client certificate chain using an external proxy path certificate validation algorithm [1].
– Ultimately, if this fails too, the method raises a standard validation exception, indicating that the client certificate was not validated.

The external proxy path validation is realized by the `ProxyPathValidation` class provided by the COG-JGlobus [2] project [8]. The supplied "validate" method is able to handle rfc3820-based proxy certificates, as well as the complete range of the old-style legacy Globus proxy certificates.[3] The scheme is easily integrated to the existing gateway classes and specifically the `ListenerManager` class that handles the incoming connections, as an inner class. Several jar bundles have to be included to the gateway libraries, mainly the cog-jglobus.jar (that actually provides the `ProxyPathCertificate` class) along with some other jar files that in turn JGlobus depends on (puretls, cryptix, etc.)

Figure 1 depicts how a normal client interaction operation occurs and how an operation using proxy certificates takes place.

4 Validation Testing

A demo CA was created using OpenSSL. The enhanced gateway was configured to trust certificates issued and signed by this CA. A user certificate and a corresponding user key were issued by this demo CA. Subsequently, proxy certificates were issued based on that user certificate by OpenSSL.

It was verified that the enhanced gateway could handle all kinds of proxy-style certificates (RFC or not), by using the grid-proxy-init tool of both the Globus toolkit and the gLite in order to generate the various certificates.

The following comprehensive test cases were realized, in order to validate the correctness of the implementation using the OpenSSL client functionality with different kinds of proxy certificates, and connecting to the enhanced UNICORE gateway to verify the authentication decision:

– Connecting using a plain (non-proxy) user certificate signed by a trusted CA
– Connecting using a plain (non-proxy) user certificate signed by an non-trusted CA
– Connecting using a proxy user certificate signed by a user certificate which in turn was signed by a trusted CA
– Connecting using a proxy user certificate signed by a user certificate which in turn was signed by a non-trusted CA
– Repeated the above two test-cases using various types of proxy-style certificates
– Repeated the tests for expired proxy certificates

At the point where the tests proved successful, the functionality of the enhanced gateway was fully verified and it was asserted that proxy signed jobs could be executed. This was realized by using the UNICORE CLI to build a standard sample task (AJO) and submit it to the proxy-enabled gateway.

The above tests should cover most of in not all of the potential user cases, and were thoroughly successful.

[2] Specifically, release 1.4 of the COG package.

[3] The usual technique used in pre-RFC era proxy certificates format, was to prepend the subject line with a "CN=proxy" entry.

5 Towards Grid Middleware Interoperability

The main obstacles encountered in trying to achieve full interoperability between the various middleware software systems, are the different tokens of authentication and authorization used, as well as the specific language that each system uses to describe the task to be submitted and various restrictions and requirements specific to that. In our current efforts [2] the main interest is concentrated around the interoperability between the major middleware systems, namely UNICORE, gLite and Globus. Adding support for proxy certificate authentication in the UNICORE system with the proposed solution completes the first aspect of interoperability, as proxy certificates are the primary form of authentication as well as authorization (when used in conjunction with attribute certificate extensions [9]) tokens in the other middleware systems.

Several projects have focused on the other half of the interoperability problem, specifically the inter-mapping of the various job languages. For example, GlobusTSI [10] that made a UNICORE job submission transparently compatible with Globus NJS backends. This work is currently deprecated and not compatible with the current releases of Globus and UNICORE. Other efforts are underway to transform JSDL [11] submissions into AJO tasks and conversely [12].

From an authorization angle, the UNICORE model lacks sophisticated support, as it merely distinguishes users only by utilizing the local username mapping (Xlogin). This essentially delegates the authorization problem to the underlying operating system access control—which in the common case (UNIX, Linux) is simply the discretionary model. The authorization problem in UNICORE is double-faceted: the transport of authorization information within the authentication tokens (signed AJOs) and the actual enforcement of these pieces of information accordingly. The first part of the problem can be fairly trivially solved by enhancing the structure of the proxy certificate, adding attribute information. This scheme has been extensively used by the other middleware systems by embedding various pseudo-attribute extensions to the proxy certificate thus augmenting it to include authorization information [13]. A standardization effort is underway, attempting to formally define a structure for these extensions that have been described by an RFC [9]. Currently, not many software implementations exist, but various endeavors are on-going [14,15].

The second part of the problem, the utilization of the authorization information, could be directly solved by making use of additional features of the underlying operating system (e.g. UNIX user groups), in support of policy enforcement. More elaborate operating system setups, such as those supporting filesystem extended attributes and full-fledged access control lists [16] can be leveraged to fully utilize the supplied authorization information.

Overall, proxy profiles adopt quite well as temporary authentication tokens to convey authorization related information, as the valid life-time of the later is usually very limited, a fact that suites the commonly small validity duration period of the proxy certificates in contrast to plain X.509 that have a much longer lifetime (6-12 months).

6 Alternative Approaches and Solutions

6.1 Explicit Trust Delegation

Recognizing the need for privilege delegation due to the very dynamic nature of Grids, the UNICORE/FUJITSU team initially proposed an alternative to using proxy certificate profiles, a scheme known as ETD (Explicit Trust Delegation) [17]. This was made to avoid using the then non-standard and not very well supported proxy extensions, which had very few implementations, and while having in mind a trade-off between flexibility and security of the two different solutions.

The proposed solution was to enhance the UNICORE architectural model at its core, introducing an explicit role model to assist in stricter privilege separation. In this scheme, a new role is added, the *User role*, splitting the authority of the legacy *Endorser role* in two separate parts. The end-user retains the User role for creating jobs, while the *Endorser role* is only retained for the authorization action. The *Consignor role*, is typically held by either the end-user to transfer the signed AJO to the server, or by a UNICORE server transporting a sub-AJO to another server. Thus, the user commonly assumes all three roles (User, Endorser and Consignor) for submitting jobs. The servers have the ability to not only consign jobs, but endorse them on behalf of the user, achieving a strict form of trust delegation with the restriction that only the end-user can act in the User role and create jobs.

Although overall this scheme is efficient and secure, it currently also excludes the desired interoperability property that the proxy certificates provide.

6.2 The CONDOR-UNICORE Bridge

Condor [18] is a job queuing/scheduling system aiming at supporting high-throughput computing by providing a sophisticated workload management system. It supports functionality for smart matching of requested resources with available resources. As Condor attempts to be quite flexible as a system, it can be used as a component for building larger systems comprised of numerous distributed Condor installations as well as other Grid-like environments.

Condor has mostly been supporting submission of jobs to foreign Grid engines, mainly the various Globus Toolkit releases (GT2+). Recently, the Condor team has introduced a new model for easy integration of job submission "backends" requiring no modifications on the main Condor sources. A proof-of-concept implementation of this new framework was the Condor-UNICORE Bridge [19]. This approach has the advantage that it

– smoothly integrates within existing deployments. No source code or any other kind of configuration modifications are required in order for this to work with UNICORE.

The disadvantages are that it

– introduces one additional software layer. Installation, configuration, maintenance, and user training overhead.

– No source code availability (this is conditional).
– Insecurity: the user keystore passphrase is kept in file to avoid repeated user interaction in cases of failure (for recovery). A solution for this might be the usage of RFC3820 proxy X.509 certificates.

The Condor-UNICORE Bridge approach as a credential provisioning solution might be a proper one when there is already a Condor deployment which is used by users as primary means for submitting any kind of tasks to systems. The GAHP [19] provides a smooth integration interface for extending the interoperability of the Condor system with other systems. This does not offer any kind of limited and restricted authentication and authorization capabilities in contrast to the proxy certificate profile approach. The Condor-G project [20], incorporated more extensive support for job submission to Globus sites into Condor. It has also added the ability to make use of MyProxy [21] for credential management, although for proxy certificates to be used for authentication in UNICORE, the presented enhancement—or a functionally similar solution—has to be available.

7 Conclusion

UNICORE is a very mature grid software system, with a wide deployment base. With the ever-increasing grid site installations and different grid middleware systems, a concern of imperative importance is the interoperability between heterogeneous software environments. This paper presented a brief overview of the traditional UNICORE authentication and authorization model, as well as the main barriers that exist preventing the harmonic symbiosis and compatibility of the various grid systems. A simple solution was introduced, extending UNICORE to support authentication using a proxy certificate profile extension. The proposed solution provides a significant step towards the near-term interoperability goals, which are the main concern of the OMII-Europe project.

Acknowledgement

This work is supported by the European Commission through the OMII-Europe project-INFSO-RI-031844. For further information please refer to [2].

References

1. Housley, R., Polk, W., Ford, W., Solo, D.: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile (2002), `http://www.ietf.org/rfc/rfc3820.txt`
2. Open Middleware Infrastructure Institute for Europe. Project no: RI031844–OMII-Europe, `http://omii-europe.org`
3. UNICORE: release 5, `http://www.unicore.eu`
4. Romberg, M.: The unicore architecture: Seamless access to distributed resources. In: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-1999) (1999)

5. OpenSSL, http://www.openssl.org/
6. Java Secure Socket Extension (JSSE) Reference Guide for the Java 2 SDK, Standard Edition, v 1.4.2, http://java.sun.com/products/jsse/reference/docs
7. Castle, B.: Java crypto api, http://www.bouncycastle.org/
8. The Java Commodity Grid Kit (v1.4). http://www-unix.globus.org/cog/distribution/1.4/api/index.html
9. Farrell, S., Housley, R.: An internet attribute certificate profile for authorization, http://www.ietf.org/rfc/rfc3281.txt
10. Riedel, M., Mallmann, D.: Standardization processes of the unicore grid system. In: Proceedings of 1st Austrian Grid Symposium, pp. 191–203. Austrian Computer Society, Schloss Hagenberg, Austria (2005), http://www.fz-juelich.de/zam/vsgc/pub/riedel-2006-SPU.pdf
11. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., Savva, A.: Job submission description language (jsdl) specification, version 1.0., http://www.ogf.org/documents/GFD.56.pdf
12. Russel, M., et al.: Transformation of JSDL into UNICORE AJOs. Poznan Supercomputing & Network Center. Personal Communication (March 2007)
13. Ciaschini, V.: A VOMS Attribute Certificate Profile for Authorization (2007), http://egee-jra-data.web.cern.ch/egee-jra1-data/glite-stable/stage/share/doc/voms/AC-RFC.pdf
14. Montes, J.A.M., Bernal, F.M., Sanchez, J.M.R.: The OpenPMI project: OpenSSL+AC, http://openpmi.sourceforge.net/
15. Levitte, R.: Official Support of Attribute Certificate Profiles in OpenSSL. Private communication (May 2007)
16. Authors, V.: Linux extended attributes and acls, http://acl.bestbits.at/
17. Snelling, D., van de Berge, S., Li, V.: Explicit trust delegation: Security for dynamic grids. FUJITSU Scientific and Technical Journal 40(2), 282–294 (2004)
18. Condor, High Throughput Computing Project, http://www.cs.wisc.edu/condor/
19. Nakada, H., Yamada, M., Itou, Y., Nakano, Y., Matsuoka, S., Frey, J.: Design and Implementation of Condor-UNICORE Bridge
20. The Condor-G Project, http://www.cs.wisc.edu/condor/condorg/
21. Basney, J., Humphrey, M., Welch, V.: The MyProxy Online Credential Repository. Software: Practice and Experience 35(9), 801–816 (2005)

Using SAML-Based VOMS for Authorization within Web Services-Based UNICORE Grids

Valerio Venturi[1], Morris Riedel[2], Shiraz Memon[2], Shahbaz Memon[2], Federico Stagni[1], Bernd Schuller[2], Daniel Mallmann[2], Bastian Tweddell[2], Alberto Gianoli[1], Sven van den Berghe[3], David Snelling[3], and Achim Streit[2]

[1] National Institute of Nuclear Physics (INFN), Bologna, Italy
[2] Forschungszentrum Juelich (FZJ), Juelich, Germany
[3] Fujitsu Laboratories of Europe (FLE), London, UK

Abstract. In recent years, the Virtual Organization Membership Service (VOMS) emerged within Grid infrastructures providing dynamic, fine-grained, access control needed to enable resource sharing across Virtual Organization (VOs). VOMS allows to manage authorization information in a VO scope to enforce agreements established between VOs and resource owners. VOMS is used for authorization in the EGEE and OSG infrastructures and is a core component of the respective middleware stacks gLite and VDT. While a module for supporting VOMS is also available as part of the authorization service of the Globus Toolkit, there is currently no support for VO-level authorization within the new Web services-based UNICORE 6. This paper describes the evolution of VOMS towards an open standard compliant service based on the Security Assertion Markup Language (SAML), which in turn provides mechanisms to fill the VO-level authorization service gap within Web service-based UNICORE Grids. In addition, the SAML-based VOMS allows for cross middleware VO management through open standards.

1 Introduction

The concept of Virtual Organization (VO), defined as a dynamic collection of individuals, institutions, and resources, emerged as central within world-wide Grid and e-Science infrastructures that deal with the so-called 'Grid problem': flexible, secure, coordinated resource sharing across dynamic, multi-institutional collaborations [1]. Enabling VOs implies requirements for a highly dynamical fine grained access control over shared resources. Resource owners makes agreements with the VO on sharing their resources provided they have control on how the sharing is done.

Enabling VO management means providing instruments to facilitate the enforcement of such agreements. One such instrument is the Virtual Organization Membership Service (VOMS) [2]. VOMS is an Attribute Authority (AA) focused on VO management. It allows VO manager to assign attributes to users according to their position in a VO. With position we mean group or project membership, role possession, or generic key value pair attributes. On request VOMS releases signed assertions containing the above described attributes. These attributes are used at the resource level to drive authorization decisions. Thus VOMS supports dynamic, fine-grained access control needed to enable resource sharing across VOs.

VOMS is used for authorization in the Enabling Grid for E-SciencE (EGEE) [3] and Open Science Grid (OSG) [4] Grid infrastructures and is thus a core component of the respective middleware stacks, the Lightweight Middleware for Grid Computing (gLite) and the Virtual Data ToolKit (VDT). In addition, a module for supporting VOMS is also available as part of the Globus Toolkit (GT) [5] authorization framework[1]. Hence, all these Grid middleware systems as well as production Grids provide sophisticated VO-level authorization. This can be significantly improved within UNICORE [6] Grids such as DEISA [7]. The absence of VO-level fine-grained authorization within UNICORE motivates our work to support VOMS authorization within the new Web services-based UNICORE 6 Grid middleware. The benefits are two fold. First, it fills the gap of having a VO-level authorization available within UNICORE Grids. Second, it lays the foundation for interoperability with other VOMS-based Grid infrastructures in terms of security and VO management.

This paper describes our work to make VOMS available under UNICORE 6 as it is developed within the Open Middleware Infrastructure Institute for Europe (OMII-Europe) [8]. The paper introduces VOMS and decribe its evolution towards an open standard compliant version based on the Security Assertion Markup Language (SAML) [9]. The core of the paper describes the usage of signed SAML assertions released by the VOMS server with the Web services-based UNICORE 6 Grid middleware. In details, it will describe how the assertions are transported in the Simple Object Access Protocol (SOAP) headers of message exchanges between UNICORE and its clients. In addition, the paper explains how the Extensible Access Control Markup Language (XACML) [10] of OASIS can be used in conjunction with VOMS and its SAML-based assertions to realize fine-grained authorization decisions within UNICORE.

The remainder of this paper is structured as follows. Section 2 presents the evolution of VOMS toward open standards and its adoption of SAML. How UNICORE 6 is capable of using VOMS as AA is described in Section 3, and in Section 4 we provide a use case scenario of how this newly developed VOMS support in UNICORE can be used by higher-level services. Related work is addressed in Section 5. The paper closes with some concluding remarks.

2 Evolution of Virtual Organization Membership Services

VOMS was originally developed in the framework of the European Data Grid and DataTag collaborations to solve the problem of granting Grid users authorization to access resources within a VO scope, enabling the high fine-grained, complex level of access control needed for sharing agreements in a VO. VOMS allows services to drive authorization decisions based on the position of the user in a VO.

[1] http://dev.globus.org/wiki/VOMS

In the last years, VOMS has been developed within the EGEE project. It is a basic component of the EGEE Grid middleware, gLite and it is used in many Grids world-wide (OSG, D-Grid, NAREGI). A module for using VOMS for authorization is available with the Globus Toolkit.

In more detail, VOMS is a system for managing the user's membership and position in a VO, such for example as group membership and role possession. These information are used by Grid services to allow for access control. VOMS has a Web-based administrative interface as well as command-line tools to administer the VO: add and delete users, create groups, assign roles or other attributes. The core component is an AA that releases signed assertions containing user's attributes holding the position of the user in the VO. In the server that is used in production today, these attribute are carried in Attribute Certificates (ACs) compliant to RFC 3821 [11]. In the most adopted usage pattern, the AC is inserted in an extension of proxy certificates of the users, as shown in 1(a). When the user authenticates with the Grid services, the services extract the AC from the proxy and authorize the access to the resource based on the these attributes.

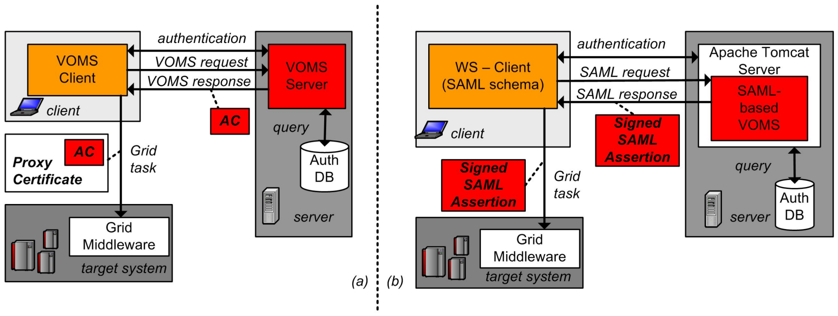


Fig. 1. VOMS server releases Attribute Certificates (a) or signed SAML assertions (b)

Within the OMII-Europe project, the VOMS server is being re-engineered to support standards emerging from the Grid community. The OGSA Authorization Working Group of the Open Grid Forum (OGF) has been working on recommendations for standardization of authorization related services. Following the work in the OMII-Europe project, the working group is defining a specification for a profile for attributes retrieval services based on OASIS SAML. The VOMS SAML Service implements that specification. The aim of the standardization activity in the OMII-Europe project is to have a VO management solution across different middleware distributions. The VO management services is part of a wider activity of re-engineering by the OMII-Europe project that aims at interoperability across of selected basic component of different grid middlewares such as UNICORE, gLite and GT. Components being re-engineered are for example Job Submission services (OGSA-BES [12]) and Data Access services (OGSA-DAI [13]).

The VOMS SAML Service is a Web service implementing protocols and binding defined in the SAML set of specifications [14] [15]. A prototype is available on the OMII-Europe Evaluation Infrastructure. The VOMS SAML Service uses Axis as SOAP implementation and can be deployed in a service container such as Tomcat. The package does not provide an API, with the aim of letting each consumer use their preferred Web services tools. The distribution comes with examples of consuming the service using popular SOAP implementations such as Axis, XFire, gSOAP, ZSI.

3 Using SAML-Based VOMS with UNICORE 6

VOMS is used within gLite (EGEE), VDT (OSG), and GT (TeraGrid), mainly by using RFC 3821 compliant ACs within a proxy certificate of end-users. This usege pattern makes VOMS widely adopted with middlewares that uses proxy

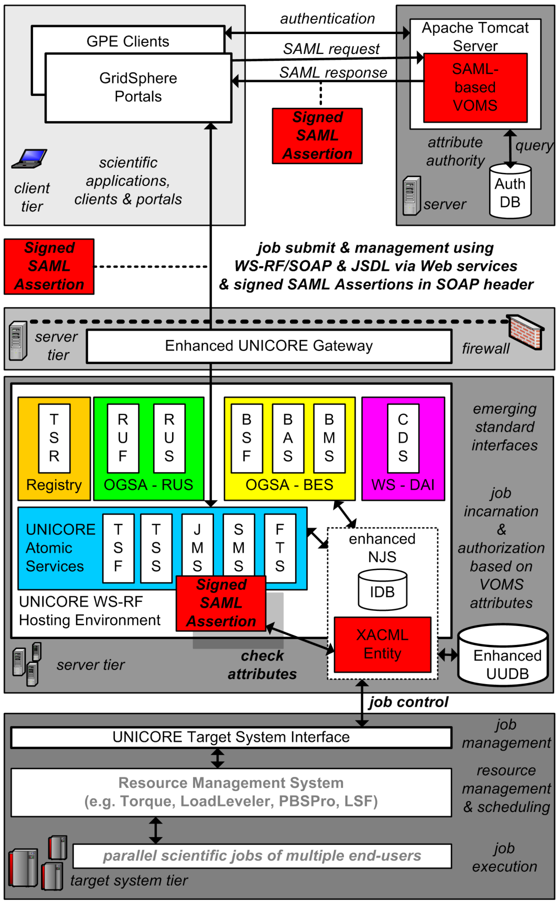


Fig. 2. VOMS acts as a AA for UNICORE 6 and releases signed SAML assertions with VOMS attributes. Based on these attributes, XACML policies are used to realize authorization decisions within the UNICORE 6 backend.

certificates, but not as widely adopted in middlewares using end-entity certificates such as UNICORE. Therefore, it becomes very helpful that the newly developed VOMS server is releasing SAML-compliant assertions that are independent from proxy certificates and thus can be used in environments that use end entity certificates. This paragraph describes how this is done in a way that VOMS can be used as AA for UNICORE Grids, including details about the authorization decisions based on VOMS attributes as shown in Figure 2.

3.1 VOMS as Attribute Authority for UNICORE

Figure 2 shows how we have used VOMS as an AA for the Web service-based UNICORE 6 Grid middleware. The client retrieve a SAML assertion from the VOMS SAML Service. While proxy certificates proved a very effective way of

```
<soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 <wsa:To>http://...services/BESFactory?res=defaultbesfactory</wsa:To>
 <wsa:Action>http://.../BESFactoryPortType/CreateActivity</wsa:Action>
 <wsa:MessageID>...CBA4</wsa:MessageID>
 <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   ID="_ec36fa7c396ka4nqa91jst"
   IssueInstant="2007-04-22T14:34:10.059Z"
   Version="2.0"
  <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
   CN=omii002.cnaf.infn.it,L=CNAF,OU=Host,O=INFN,C=IT
  </saml:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   ...
  </ds:Signature>
  <saml:Subject>
   <saml:NameID
     Format="urn:oasis:names:tc:SAML:1.1:nameid-format:x509SubjectName">
    CN=Morris Riedel,OU=ZAM,OU=Forschungszentrum JuelichGmbH,
    O=GridGermany,C=DE
   </saml:NameID>
  </saml:Subject>
  <saml:SubjectConfirmation
    Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
   ...
  </saml:SubjectConfirmation>
  <saml:Conditions NotBefore="..." NotOnOrAfter="..." />
  <saml:AttributeStatement>
   ...
  </saml:AttributeStatement>
 </saml:Assertion>
</soap:Header>
```

Fig. 3. SAML-based assertions released from the VOMS server are transferred within the SOAP header of Web service message exchanges between UNICORE and its clients

```
<saml:Assertion>
 ...
 <saml:AttributeStatement>}
  <saml:Attribute Name="group-membership-id" NameFormat="urn...">
   <saml:AttributeValue type="xs:string">
    /omiieurope
   </saml:AttributeValue>
  </saml:Attribute>
 </saml:AttributeStatement>
 ...
</saml:Assertion>
```

Fig. 4. Simple example of an VOMS FQAN (group-membership-id with value omiieurope) encoded in an saml:AttributeStatement element within a saml:Assertion element. In this context an XACML policy can be defined within UNICORE that only allow members of the omii-europe group to access the requested service.

tranporting ACs, we found that the most natural way of transporting SAML assertions is in the SOAP Header of Web services messages as shown in Figure 3. A similar mechanism is described in the OASIS Web Services Security set of specification.

3.2 Authorization Decisions in UNICORE Based on XACML

Figure 2 also shows how SAML assertions from the VOMS are used during authorization decisions in conjunction with XACML policies. UNICORE 6 incorporates a Policy Decision Point (PDP) that uses Extensible Access Control Markup Language (XACML) policies during authorization decisions in conjunction with the UNICORE User Database (UUDB). In the context of VOMS, these XACML policies can be used to make attribute-based authorization decisions based on SAML assertions released from the VOMS server and that are transported to UNICORE within the SOAP header. Part of these assertions are *saml:AttributeStatement* elements that provide values for Fully Qualified Attribute Names (FQANs) stating role possession or group/project membership as shown in Figure 4. Finally, the SAML-based assertions are signed with the certificate of the VOMS server, which can be verified at the resource level to check if the assertion comes from a trusted VOMS server.

4 Use Case Scenario: Role-Based Authorization in Collaborative Visualization and Steering Sessions

UNICORE 6 is easily extensible and allows for the development of higher-level services that work on top of the UNICORE Atomic Services (UAS) [16]. One of these services is the Collaborative Online Visualization and Steering (COVS) [17] Grid service that allow multiple participants sharing the same visualization session. COVS sessions support multiple roles for users and thus is a good proof of concept of how VOMS can be used for role-based authorization in UNICORE 6.

The master role is the initiator of a COVS session and is able to configure different setups during a session that can not be done by usual participants. A person that acts in this role uses the Grid middleware client and its COVS plugin to access the COVSFactory service. The COVSFactory service creates in turn a COVS session resource, which includes the startup of components realizing the communication between a parallel simulation on supercomputers or clusters and visualization clients. The participant role on the other hand is defined as any person that participates in a COVS session, which also includes the person in the master role. End-users that want to participate also use the Grid middleware client to join a COVS sessions, but are not allowed to create a session.

In this context, VOMS provides fine-grained authorization based on different roles or on group memberships. Therefore it can be used to control which users are able or not able to participate (e.g. a research group named astro-d configured within the VOMS server is allowed to share the view of a visualization or normal participants are usually not allowed to submit computational jobs). The VOMS server can release signed SAML assertions with role attributes checked at service-level at the COVSFactory and COVSSession service as shown in Figure 5.

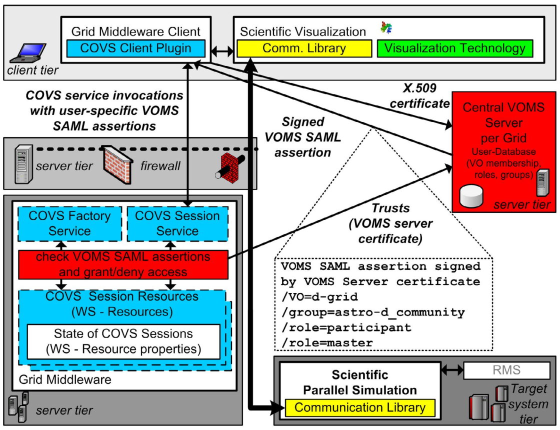

Grid Middleware Client
COVS Client Plugin
Scientific Visualization
Comm. Library
Visualization Technology
client tier
COVS service invocations with user-specific VOMS SAML assertions
X.509 certificate
Signed VOMS SAML assertion
Central VOMS Server per Grid
User-Database (VO membership, roles, groups)
server tier
firewall
COVS Factory Service
COVS Session Service
check VOMS SAML assertions and grant/deny access
COVS Session Resources (WS - Resources)
State of COVS Sessions (WS - Resource properties)
Grid Middleware
Trusts (VOMS server certificate)
VOMS SAML assertion signed by VOMS Server certificate
/VO=d-grid
/group=astro-d_community
/role=participant
/role=master
Scientific Parallel Simulation
Communication Library
RMS
Target system tier

Fig. 5. Using the SAML-based VOMS to manage different roles in COVS sessions

5 Related Work

The other main AA within Grids is Shibboleth, a tool that provides a federated single sign-on and attribute exchange framework, mainly used in the education community. GridShib is a software product that allows for interoperability between GT and Shibboleth, thus making the latter available for Grid authorization. GridShib project members are working on an OASIS standard for a deployment profile for X.509 subject to use with SAML 2.0. This profile complements the SAML specifications with indications on how to use SAML with X.509 certificates. Therefore, members from the GridShib and VOMS teams have agreed within the OGF OGSA Authorization WG that this specification in combination with SAML 2.0 is the specification for AA services. To sum up, it is expected that GridShib and VOMS follow the same standard interface for message exchanges as well as assertions and thus the work describes in this paper should work not only with VOMS but also with GridShib, except the different policy definitions that are dependent from the correspondent attribute formats. In addition, VOMS and Shibboleth use the same Internet2 OpenSAML toolkit source code. VOs and authorization based on information about the role of users within VOs are missing concepts in production UNICORE 5 today. UNICORE 5 also lacks support for using attributes of a user retrieved from his home institution. Overcoming these limitations is part of the IVOM project [18] that is part of the German D-Grid Initiative [19]. In contrast to our work that rely on UNICORE 6 and the SAML-based VOMS server, the IVOM project develops solutions for UNICORE 5 and the Attribute Certificate-based VOMS server.

6 Conclusions

An initial prototype of using the SAML-based VOMS in conjunction with UNICORE 6 was shown at OGF20 at the OMII - Europe booth and is currently further improved within the evaluation infrastructure of OMII-Europe.

This in particular fills the gap of UNICORE's limited VO-level authorization functionality since the new SAML-based VOMS server is independent from gLite and thus can be used by purely UNICORE Grids as Attribute Authority.

We have further shown that a VO is now capable to offer its end-users both high throughput computing resources (nodes in a farm running gLite) and high performance resources (supercomputers running UNICORE) if the middleware provide services that are compliant to emerging standards interfaces such as OGSA-BES or WS-DAIS. In other words, this work has a significant impact regarding the interoperability between UNICORE and gLite primary, but also Globus can be configured to use VOMS and thus VOMS can act as an AA for VOs to use cross-middleware Grid resources.

Finally, even if interoperability is technically possible, the adoption of these new developed components within production e-Infrastructures such as DEISA or EGEE is rather slow. However, it is expected that UNICORE 6 and thus the support for the new SAML-based VOMS server will considered as the next production version within these e-Infrastructures.

Acknowledgements

This work is partially funded by the OMII-Europe project under EC grant RIO31844-OMII-EUROPE, duration May 2006 - April 2008.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications 15(3) (2001)
2. Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Frohner, Á., Lörentey, K., Spataro, F.: From gridmap-file to voms: managing authorization in a grid environment. Future Generation Comp. Syst. 21(4), 549–558 (2005)
3. Enabling Grid for E-sciencE, `http://www.eu-egee.org/`
4. Open Science Grid, `http://www.opensciencegrid.org/`
5. The Globus Toolkit, `http://www.globus.org/toolkit`
6. Streit, A., Erwin, D., Lippert, T., Mallmann, D., Menday, R., Rambadt, M., Riedel, M., Romberg, M., Schuller, B., Wieder, P.: UNICORE - From Project Results to Production Grids. In: Grandinetti, L. (ed.) Grid Comp. and New Frontiers of High Performance Proc., pp. 357–376. Elsevier, Amsterdam (2005)
7. DEISA - Distributed European Infrastructure for Supercomputing Applications, `http://www.deisa.org`
8. The Open Middleware Infrastructure Institute for Europe, `http://omii-europe.org/OMII-Europe/`
9. OASIS Security Services (SAML) TC, `http://www.oasis-open.org/committees/security`
10. OASIS eXtensible Access Control Markup Language (XACML) TC, `http://www.oasis-open.org/committees/xacml`
11. S.Farrell, R.: An Internet Attribute Certificate Profile for Authorization (2002), `http://www.ietf.org/rfc/rfc3281.txt`
12. OGSA Basic Execution Services WG, `http://forge.gridforum.org/projects/ogsa-bes-wg`
13. Database Access and Integration Services (DAIS), `https://forge.gridforum.org/sf/go/proj1070`
14. Cantor, S., et al.: Assertions and Protocols for the Security Assertion Markup Language (SAML) Vol 2, (2005) `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`
15. Cantor, S., et al.: Bindings for the Security Assertion markup Language (SAML), vol. 2 (2005), `http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf`
16. Riedel, M., Mallmann, D.: Standardization Processes of the UNICORE Grid System. In: Proceedings of 1st Austrian Grid Symposium 2005, pp. 191–203. Austrian Computer Society, Schloss Hagenberg, Austria (2005)
17. Riedel, M., Eickermann, T., Frings, W., Dominiczak, S., Mallmann, D., Dssel, T., Streit, A., Gibbon, P., Wolf, F., Schiffmann, W., Lippert, T.: Design and evaluation of a collaborative online visualization and steering framework implementation for computational grids. In: Proc. of the 8th IEEE/ACM International Conference on Grid Computing (Grid 2007), Austin, Texas (to appear)
18. Interoperability and Integration of VO-Management Technologies in D-Grid, `http://www.d-grid.de/index.php?id=314\&L=1`
19. D-Grid Initiative, `http://www.d-grid.de/index.php?id=1\&L=1`

Attributes and VOs: Extending the UNICORE Authorisation Capabilities

Arash Faroughi[1], Roozbeh Faroughi[1], Philipp Wieder[2], and Wolfgang Ziegler[1]

[1] Fraunhofer Institute SCAI, Department of Bioinformatics,
53754 Sankt Augustin, Germany
{arash.faroughi,roozbeh.faroughi,wolfgang.ziegler}@scai.fraunhofer.de
[2] Central Institute for Applied Mathematics, Research Centre Jülich,
52425 Jülich, Germany
ph.wieder@fz-juelich.de

Abstract. Reliable authentication and authorisation are crucial for both service providers and their customers, where the former want to protect their resources from unauthorised access and fraudulent use while their customers want to be sure unauthorised access to their data is prevented. In Grid environments Virtual Organisations (VO) have been adopted as a means to organise and control access to resources and data based on roles that are assigned to users. Moreover, attribute based authorisation has emerged providing a decentralised approach with better scalability. Up to now UNICORE authentication and authorisation is based on X.509 certificates only. In this paper we will present two approaches to integrate both role or attribute based authorisation using VOMS and attribute based authorisation using Shibboleth into UNICORE.

1 Introduction

In collaborative distributed environments like Grids where services are offered by multiple service providers and consumed by a large number of clients, the concept of linking authorisation to roles and attributes may help lowering the complexity of user management and user authorisation while making the process more transparent. In Grid environments the concept of Virtual Organisations (VOs) [8] are utilised as a powerful instrument for creating dynamic organisations whose members are sharing a common goal, e.g. researchers working together on a dedicated subject for a limited time using shared resources. Based on roles defined for VO members the service provider will enforce the access control for resources and data. A well known example are the four major experiments around the Large Hadron Collider [10] where scientists are organised in four corresponding VOs to share their data and get access to the computational and storage resources of EGEE [4]. In general, the VO-based approaches are relying on users being furnished with X.509 certificates from a trusted Certificate Authority (CA) operated e.g. by their home institution or on a project level. However, setting up such trusted Public Key Infrastructures (PKI) is often beyond the capabilities of smaller institutions or regarded as imposing too much effort for large institutions with a huge number of certificates to be managed.

In collaborative environments without a PKI the concept of identity federation has been identified as a mechanism for authentication and authorisation. The most prominent system is Shibboleth [19]. Shibboleth implements a distributed approach where the user authenticates vis--vis his home institution when trying to access resources of a service provider and in turn the service provider gets access to selected attributes of this user, which are maintained by his home institution. Based on these attributes the service provider then decides on the authorisation of the user to access the requested resources.

For the rest of this paper we will use the term role-based authorisation for authorisation based on the information about a user within a VO maintained by a VO Management system and usually included in his X.509 certificate like e.g. when using the Virtual Organisation Membership Service (VOMS) [21]. In contrast, we will use the term attribute-based authorisation when attributes of a user stored at his home institution are used for taking authorisation decisions, i.e. when using Shibboleth mechanisms to retrieve these attributes.

The authentication and authorisation in UNICORE is based on plain X.509 certificates until now, which implies several limitations, e.g. single sign on (SSO) for Grid resources that are not part of a UNICORE environment is not supported as UNICORE so far only allows static explicit trust delegation (ETD) and users without a certificate could not access UNICORE resources of a UNICORE based Grid at all. Meanwhile, some of the issues were smoothed out by the GRIP project [11] and trust delegation through proxy-certificates is supported by UNICORE. However, VOs and authorisation based on information about the role of users within VOs are missing concepts in UNICORE today. UNICORE lacks also support for using attributes of a user retrieved from his home institution. Overcoming these limitations is part of the IVOM project [9] funded by the German D-Grid Initiative [3] and we will present work done in IVOM in this paper.

The remainder of the paper is organised as follows: Section 2 describes current attribute-based authorisation work. Section 3 gives a brief overview on the two basic technologies for role-based and attribute based authorisation we consider for the integration with UNICORE. Section 4 introduces the current UNICORE mechanisms for authentication and authorisation. The architectures for the integration with VOMS and Shibboleth are presented in Section 5. Section 6 describes future work and concludes the paper.

2 Related Work

The Swiss National Research Network (NREN) SWITCH is setting up a Shibboleth based authentication and authorisation infrastructure (AAI) for the Swiss Research infrastructure. SWITCH has set up a federation of Swiss Identity Providers (IdP) and Service Providers (SP) and operates the necessary Short Lived Credential Service (SLCS) [15]. SWITCH also developed the VOMS

Attribute from Shibboleth (VASH) [7] service that allows transferring the Shibboleth user attributes into VOMS. At the time of the VASH development GridShib was not considered because of its limitation to the pull-model for retrieving the attributes from the IdP. The German NREN DFN also has almost completed the setup of a federation of German IdPs and SPs [2]. DFN is finalising the accreditation of its SLCS by the EUGridPMA and is expected to go into productive operation in near future. More work on integration of the different existing VO-Management technologies and the introduction of attribute-based authorisation in D-Grid is done in the IVOM project [9]. Finally the OMII-Europe project [12] is working on integration of SAML assertions into UNICORE coming from a modified VOMS based on the emerging OGSA AuthZ standard.

3 Role-Based and Attribute-Based Authorisation

3.1 Virtual Organization Membership Service (VOMS)

VOMS has been developed as part of the joint efforts of the European DataGrid and DataTAG projects. It is a system, which classifies users that are participating in a VO based on a set of attributes that will be granted to them upon request. These attributes will be included into Globus-compatible proxy-certificates for supporting SSO in grid-environments. VOMS consists of four main components [1]:

User Server: receives requests from a client and returning information about the user

User Client: contacts the User Server with the user's certificate, authenticates the user to the server and creates a proxy certificate with VO Fully Qualified Attribute Name (FQAN) extensions.

Administration Client: used by the VO Administrator to add/delete and change VO-Attributes like roles and groups.

Administration Server: accepts requests from client to update the database.

Prior to getting access to the Grid the user must execute the voms-proxy-init to generate a proxy-certificate, similar to the grid-proxy-init command of the Globus Toolkit. The main difference is that VOMS includes the authorisation information

1. Using certificates the user and VOMS Server authenticate each other;
2. The user sends a request to the VOMS Server, which is signed by it;
3. The VOMS Server verifies the user's identity and checks the syntax of the request;
4. The VOMS Server sends the required authorisation information as an attribute certificate back to the user;
5. The user validates the response of the server;
6. Optionally, the user repeats this process for other VOMSes;
7. The user creates the proxy certificate and inserts the received authorisation information into a (non-critical) extension of the proxy-certificate.
8. The user may add user-supplied authentication information (e.g. Kerberos tickets)

List 1. VOMS operations

of the user into the proxy certificate retrieved from the VOMS-Server [6] resulting in an Attribute Certificate (RFC 3281). List 1 describes the procedure:

3.2 Shibboleth

The open source system Shibboleth supports an Attribute Based Access Control model. Shibboleth is a federated identity management system, developed by Internet2 and supports authorisation decisions based on the attributes of the users. It uses the Security Assertion Markup Language (SAML) to implement SSO across or within organisational boundaries. The three major components of Shibboleth are [13]]:

The **Identity Provider** (IdP) is responsible for asserting authentication and authorisation information about their Shibboleth-user. The IdP is located at the home organisation of a user. The IdP has two major services, the Handle Service (HS) and the Attribute Authority (AA). The HS authenticates the user and issues an Attribute Query Handle in the form of a signed SAML response. The AA responds to requests for user-attributes by the Attribute Requester (AR) of the Service Provider.

The **Service Provider** (SP) is offering protected resources to customers. The SP decides and enforces the authorisation to access resources. The SP consumes SAML Assertions. The most important components of the SP are the Assertion Consumer Service (ACS), the Attribute Requester (AR) and the Resource Manager (RM). The ACS validates Authentication assertions from the HS of an Identity Provider. The AR is responsible to request attributes from the user's IdP. The RM makes authorisation decisions based on the user's attributes.

The **Where Are You From Service** (WAYF) may be used to establish the communication between the Service Provider and the Identity Provider of the user. It provides a mechanism for routing users from a resource to their point of login. The WAYF Service shows a list of IdPs where users have to select their IdP.

1. A user tries to access a resource, which is located at the Service Provider
2. The Service Provider needs to identify the home organisation where the user is known. Therefore the Service Provider redirects the user to the WAYF Service.
3. The WAYF service shows a list of Identity Providers.
4. The user selects his Identity Provider aka home organisation.
5. The WAYF service redirects the user to his Identity Provider.
6. His home organisation asks the user to provide his/her login credentials.
7. The user provides his/her credentials to the home organisation.
8. After a successful AuthN the Identity Provider creates a handle and forwards this to the Service Provider.
9. The Service Provider sends an attribute request to the Identity Provider of the user by sending the received handle.
10. The Attribute Authority (AA) verifies the Handle. After a successful validation the AA follows the rules of the Attribute Release Policy when deciding whether or not to release an attribute. The AA sends the released attributes to the Service Provider. The Resource Manager makes authorisation decisions based on the user's attributes.

List 2. Ten steps accessing protected resources using Shibboleth [20] [13]

List 2 described the process of the authentication and the transfer of the user attributes to the SP.

4 UNICORE Authentication and Authorisation Model

The target sites comprising a UNICORE Grid need to verify the identity and access rights of users who want to execute tasks on the target sites, and they must, in addition, verify that the tasks they receive for execution belong to the appropriate users. To achieve this, the UNICORE security model specifies the usage of permanent X.509 certificates (which are issued by a Certification Authority (CA)) to authenticate and authorise users, to authenticate UNICORE server components, and to sign jobs and software plugins [18]. In the course of this section we contemplate user authentication and user authorisation since both mechanisms are fundamental to the enhancements we present later. The user's X.509 certificate is used to provide single-sign-on in the UNICORE client. The client unlocks the user's keystore once the correct password is entered at start-up, which implies that no further password requests are demanded from the user. To authenticate the user, the client has to present the user's X.509 certificate to the UNICORE Gateway. The certificate is issued by a CA that is being trusted within a UNICORE Grid. This implies that the signer certificate of this particular CA is included in the server components of UNICORE. Please note that UNICORE neither prescribes a specific Certification Authority nor is limited to the usage of only a single CA.

To authorise users, certificates are mapped to local accounts (i.e. in general a standard UNIX uid/gid), which may be different at each site due to existing naming conventions. The rules that describe the mapping of a user's abstract identity (which is contained in the certificate) to the concrete one at the local site are contained in the UNICORE User Data Base (UUDB). Through the UUDB sites retain full control over the authorisation of users and over the underlying rules leading to the acceptance or rejection of a particular individual based on the distinguished name or other information that might be contained in the certificate. UNICORE can handle multiple user certificates (abstract identities) of a single user, i.e. it permits a client to be part of multiple, disjoint Virtual Organisations. In addition the client offers the possibility to configure different project accounts so to allow users to select different accounts for different projects on one execution system or to maintain different roles with different privileges.

The private key contained in the client keystore is also used to sign each UNICORE job (and all the nested sub-jobs). This mechanism protects against tampering while the job is transmitted over insecure connections and it allows to verify the identity of the owner at the receiving server without it trusting the intermediate sites which forwarded the job.

UNICORE, apart from extensions implemented to realise interoperation with other Grid middleware like Globus, does natively not support proxy certificates. Instead it supports an explicit trust delegation [14] that allows trusted "agents" in the Grid to create jobs on behalf of end-users. This mechanism allows services like brokers, schedulers, or third-party SLA negotiators to be integrated into a UNICORE Grid without breaching UNICORE's security model.

5 UNICORE Integration with VOMS and Shibboleth

Goals: The goals of the UNICORE integration with VOMS and Shibboleth are to extend UNICORE with attribute and role based authorisation, though to keep the modification of the UNICORE Client as minimal as possible and to keep the UNICORE authentication mechanisms as much as possible.

5.1 The Integrated Architecture of VOMS and UNICORE

To support VO-based Authorisation in UNICORE, a VO-Module is needed, which creates the user certificates with VO-authorisation information. VOMS creates credentials in form of proxy certificates. For the integration of VOMS and UNICORE a VOMS-plugin and an extension of the UUDB will be implemented:

The **VOMS-Plugin** is the VO-module for UNICORE and allows the user to specify his request for creating proxy certificates including the vo/group/role-information. The VOMS-Plugin generates a proxy certificate with VOMS-specific extensions for the end-user (by using the VOMS-command voms-proxy-init and attaches it to the Abstract Job Object (AJO) encapsulated as a site-specific security object (SSO-Object) [5].

Extended UUDB: UNICORE maps with the UUDB the identity of the end-user (DN-subject of the end-user certificate) to a local account. To support Role-Based Access Control (RBAC) or Attribute Based Access Control (ABAC) authorisation, the UUDB authorisation mechanism must be extended. For the UNICORE-VOMS integration, a new UUDB implementation is needed, which does the mapping on basis of VOMS-FQANs.

UNICORE-VOMS-Architecture: The VOMS plugin is integrated into the UNICORE client and doesn't modify the authentication mechanism of UNICORE. It extends the UNICORE client with the functionality described above. The UUDB implementation of UNICORE 5 must be replaced by the

1. The user authenticates to the UNICORE client using his permanent user certificate;
2. The user specifies his VOMS-request by using the VOMS-Plugin of the UNICORE client;
3. The VOMS-Plugin checks the syntax of the request, validates the users identity and sends the request to the VOMS Server:
4. The VOMS Server validates the request and sends the required authorisation information as an attribute certificate to the VOMS-Plugin;
5. The VOMS-Plugin generates a proxy-certificate with the received VOMS extensions.
6. The user submits a job with the usual UNICORE mechanism. The VOMS-Plugin encapsulates the proxy certificate in the AJO SSO. The UNICORE Client sends the AJO via the UNICORE Protocol through the Gateway and the NJS component.
7. The VOMS-extended UUDB maps the VO-FQAN of the user's proxy certificate to a local account.

List 3. Interaction of UNICORE and VOMS

VOMS-extended UUDB. UNICORE provides a simple way for inserting plugins and changing the UUDB-Implementation. Therefore, UNICORE can be extend with the VOMS-Implementation using standard features.

List 3 and Figure 1 illustrate the architecture of the UNICORE/VOMS integration and the communication-procedure:

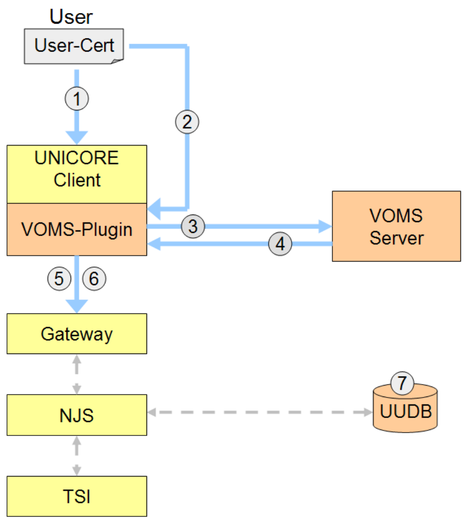


Fig. 1. Architecture of the UNICORE and VOMS integration

5.2 UNICORE Integration with Shibboleth

The major modification of UNICORE with respect to the Shibboleth integration is the UUDB, therefore the effort to integrate Shibboleth with higher UNICORE versions are expected to be small. The main tasks to extend UNICORE with shibboleth-based authorisation are:

UNICORE Authentication with a Short Lived Credential (SLC): The exchange in Shibboleth is based on assertions between an Identity Provider and a Service Provider. The AAI in UNICORE relies on the usage of X.509 certificates. With MyProxy a Shibboleth Identity can be translated to Grid Identity by generating a SLC. Using SLC the authentication mechanisms in UNICORE are largely unchanged. MyProxy issues a short lived X.509 Credentials after a successful user authentication at a Shibboleth Identity Provider. The short lived Credentials have a maximum Lifetime of 1 million seconds. To use SLC for the UNICORE user-authentication, the UNICORE Client, the UNICORE Gateway and the UNICORE NJS must trust the MyProxy Certification authority.

Attribute-Based UUDB: UNICORE maps the identity of the end-user (DN-subject of the end-user certificate) to a local account using the information stored in the UUDB. To support ABAC-Authorisation, the UUDB authorisation mechanism must be extended. For the UNICORE-Shibboleth integration, an extended UUDB implementation is needed, which does the mapping to the permissions based on the attributes.

5.3 Necessary Changes of the Shibboleth-Framework

Shibboleth's HTTP-Redirects: The usual web-based Shibboleth mechanisms of redirecting the user accessing the service provider to the WAYF server and the IdP is not practical in Grid environments submitting unattended batch jobs [17]. For this reason, the UNICORE/Shibboleth-approach does the user-authentication before requesting a protected resource. This will be realised by a module collecting the user's authentication and authorisation information for creating SLCs and convey those to the UNICORE System. The authentication and authorisation will be done in UNICORE with these SLCs.

The **WAYF Service** has a list of known and trusted Identity Providers in a Shibboleth Federation. The user first interacts directly with the WAYF service and then tries to access a resource.

UNICORE-Shibboleth-Architecture: An external Shibboleth-UNICORE module will be implemented for the creation of SLCs. The authentication is done by the IdP and the SLC is issued by MyProxy (see List 4 and Figure 2).

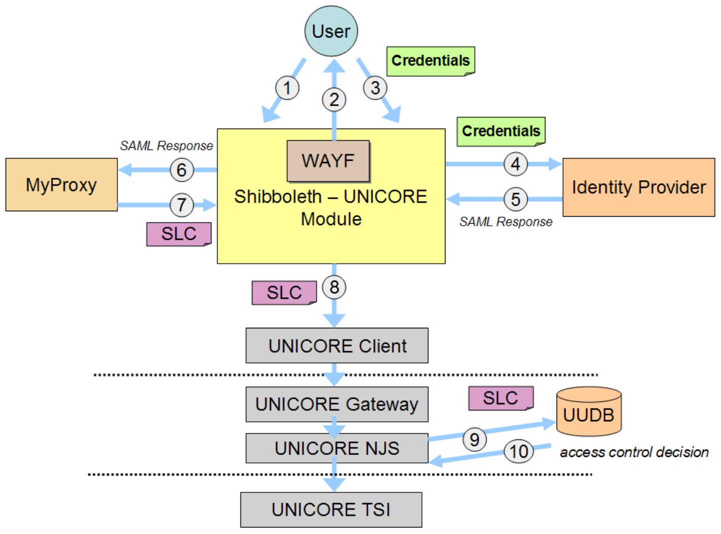


Fig. 2. Integration of UNICORE and Shibboleth

1. The user runs the Shibboleth-UNICORE Module
2. The Module fetches an up-to-date list of IdPs from the WAYF server, presents the list of Identity Providers and makes a Shib-Authentication Request.
3. The user selects his/her identity provider and gives his/her Credentials.
4. The Shibboleth-UNICORE Module authenticates the user to the Identity Provider.
5. After a successful authentication the IdP returns a SAML Response to the Shibboleth-UNICORE Module. The Response contains an authentication assertion and an attribute assertion.
6. The Shibboleth-UNICORE Module presents the Response to MyProxy.
7. MyProxy generates an X.509 credential, inserts the attribute assertion into the certificate, and returns the credential to the Shibboleth-UNICORE Module.
8. The Module runs the UNICORE Client and uses the SLC for the UNICORE user authentication.
9. The NJS sends the SLC to the UUDB.
10. Based on the attributes of the SLC the UUDB response with an access decision.

List 4. Interaction of UNICORE and Shibboleth

6 Future Work

Within the IVOM project the feasibility studies and the design of the two architectures have been completed recently and we started with the implementation of the UNICORE extensions. In the meantime the IVOM project will also provide a version of GridShib for the D-Grid which will then be integrated also allowing GridShib users accessing UNICORE resources. We also plan to integrate the VO Membership Registration Services [16] database of VO attributes with UNICORE in the next version. Also, because in D-Grid the currently used UNICORE 5 will be replaced by UNICORE 6 in 2008 we plan to integrate UNICORE 6 when it will become available. Once available we will switch to the D-Grid Short Lived Credential Service (SLCS) operated by the German DFN for the creation of SLCs. Finally, we will co-operate with the OMII-Europe project [12], which is working on similar extensions of UNICORE.

Acknowledgements

Some of the work reported in this paper is funded by the German Federal Ministry of Education and Research through the D-Grid project under grant #01AK800A. This paper also includes work carried out jointly within the CoreGRID Network of Excellence funded by the European Commission's IST programme under grant #004265.

References

1. Alfieria, R., Cecchinib, R., Ciaschinic, V., dell'Agnello, L.: From gridmap-file to voms: managing authorization in a grid environment. Future Generation Computer Systems 21(4), 549–558 (2005), `http://www.fis.unipr.it/lca/grid/doc/from-gridmap.pdf`

2. DFN-AAI: Authentication and Authorisation Infrastructure in DFN (in German) (last visited June 15, 2007) website https://www.aai.dfn.de/der-dienst.html
3. D-grid initiative (last visited June 15, 2007), website http://www.d-grid.de/index.php?id=1\&L=1
4. EGEE - Enabling Grids for E-sciencE (last visited June 15, 2007) website http://www.eu-egee.org/
5. Wieder, P., et al.: Grid interoperability project. Technical report, FZJ Jülich Germany (2002)
6. Alfieri, R., et al.: From gridmap-file to VOMS - managing authorization in a Grid environment. Technical report, INFN Parma and University of Parma (2004), http://grid-auth.info.it/docs/voms-FGCS.pdf
7. Flury, P., Tschopp, V., Lenggenhager, T., Witzig, C.: Shibboleth Interoperability with Attribute Retrieval through VOMS. Technical report, EGEE (2006), https://edms.cern.ch/file/807849/1/EGEE-II-MJRA1.5-807849-v0.95.pdf
8. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. Journal of High Performance Computing Applications 15(3), 200–222 (2001), www.globus.org/research/papers/anatomy.pdf
9. Interoperability und integration of vo-management technologies in d-grid (last visited June 15, 2007) website http://www.d-grid.de/index.php?id=314\&L=1
10. LHC - The Large Hadron Collider (last visited June 15, 2007) website: http://lhc.web.cern.ch/lhc/
11. Nicole, D.A.: UNICORE and GRIP: Experiences of Grid Middleware Development. In: Proceedings of 2005 International Conference on Grid Computing and Applications. ECS, June 2005, pp. 11–17 (2005), Online at: http://eprints.ecs.soton.ac.uk/11889/01/gca_final.pdf
12. Open Middleware Infrastructure Institute Europe - OMII-Europe (last visited June 15, 2007) website: http://omii-europe.org/OMII-Europe/
13. Scavo, T., Cantor, S.: Shibboleth architecture, technical overview. Technical report (2005), http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-tech-overview-latest.pdf
14. Snelling, D., van den Berghe, S., Li, V.: Explicit trust delegation: Security for dynamic grids. Technical report, FUJITSU Scientific and Technical Journal, 40(2), 282–294 (December 2004)
15. Short Lived Credential Service (SLCS) (last visited June 15, 2007), website http://www.switch.ch/grid/slcs/
16. VO Membership Registration Service (last visited June 15, 2007), website: http://www.uscms.org/SoftwareComputing/Grid/VO/
17. Welch, V., Barton, T., Keahey, K., Siebenlist, F.: Attributes, anonymity, and access - shibboleth and globus integration to facilitate grid collaboration (2005), Online: http://grid.ncsa.uiuc.edu/papers/gridshib-pki05-final.pdf
18. Wieder, Ph., Goss-Walter, T., Letz, R., Kentemich, T., Hoppe, H.-C.: An analysis of the unicore security model. Technical report, Global Grid Forum. Grid Forum Document - Informational 18 (GFD-I 18)
19. Shibboleth. Online: http://shibboleth.internet2.edu/
20. The Swiss Education and Research Network. Online: http://www.switch.ch/aai/demo/medium.html
21. Virtual Data Toolkit: VOMS-Documentation. Online: http://vdt.cs.wisc.edu/VOMS-documentation.html

A Business-Oriented Grid Workflow Management System

Luca Clementi[1], Claudio Cacciari[1], Maurizio Melato[2], Roger Menday[3], and Björn Hagemeier[3]

[1] CINECA,
Via Magnanelli 6/3,
40033 Casalecchio di Reno, Italy
{l.clementi,c.cacciari}@cineca.it
[2] NICE,
via Marchesi di Roero 1,
14020 Cortanze, Italy
maurizio.melato@nice-italy.com
[3] Central Institute for Applied Mathematics
Forschungszentrum Jülich GmbH
D-52425 Jülich, Germany
{r.menday,b.hagemeier}@fz-juelich.de

Abstract. The wide adoption of Service Oriented Architecture by the Grid community has made available several software tools that allow exposing hardware resources and scientific data to remote peers by means of well standardized protocols. Hence the necessity for scientists to easily design a simulation that leverages distributed applications. In this paper, we present the implementation details of A-WARE, a workflow framework that adopts recognised standards, especially by the enterprise community, like BPEL. In this way our product can boast a higher level of interoperability with other similar systems. The workflow graphical notation is also based on a standard: BPMN. BPMN provides a unique, standardized and comprehensive modeling format understandable by both business people (involved in the area of business process management) and IT people, resulting also portable across different departments or companies.

Keywords: Workflow, Grid System, SOA, BPEL, BPMN.

1 Introduction

Grid computing is enabling scientists to use heterogeneous and distributed resources and applications in a seamless way, allowing their composition into more complex tasks in a transparent manner and hiding all complexity of the underlying infrastructure.

In the early days, Grid systems were built with ad-hoc components and technologies developed specifically for particular projects sometimes using specific communication protocols. On the other hand, the recent past has seen the emerging paradigm of Service Oriented Architectures (SOA) with well standardized protocols gaining more acceptance and adoption by the Grid community [1]. The term SOA refers to systems structured as network of loosely coupled communication services [2]. To these purposes a set of technologies called Web Services (WS) have gained wide acceptance, creating a huge collection of open-source and commercial tools that support their development and deployment.

Regarding the Grid community, there are plenty of production-level products available today that allow exposing hardware and software resources using Web Services standard interface description (Globus, EGEE, UNICORE, etc.). Moreover, several scientific data repositories are currently available by means of Web Services interfaces [3], [4]. In this scenario, where there is an increasing number of distributed data sources and services exposed through WS standards, applications able to orchestrate remote WS resources with user friendly interfaces play a very important role.

Workflow management systems able to operate in a WS-standard SOA environment, would allow scientists to concentrate on the creation of their simulations and analyses, designing workflows that would take full advantage of such data sources, applications, and hardware resources. This service composition into more complex tasks would be accomplished without having to deal with the complexity of the actual service and infrastructure implementations.

The scientific community has developed various workflow systems that allow the orchestration of such a distributed resource set [5]. We believe a more standardized approach should be used to build such a framework. In this direction there is a growing interest and adoption by both scientific and business communities of Business Process Execution Language (BPEL) as a standard language to describe and execute workflows. In fact BPEL, besides being widely accepted as industry standard for defining business processes and adopted by many commercial Enterprise Application vendors, has also recently become a OASIS standard [9]. Lately, also the Grid community began to follow this standard with more interest as a possible foundation for Grid workflow systems [5].

However, BPEL does not provide any specification regarding how a process should be represented graphically. Therefore, many software vendors supporting BPEL have invented their own notation for workflows, while only few have adopted an already existing standard, the Business Process Modeling Notation (BPMN), a graphical notation for business processes developed by the Object Management Group [8].

In this paper we present the implementation of a workflow infrastructure system, which is primarily based on BPEL as a runtime language and on BPMN for the graphical representation, highlighting the most challenging problems that arise from our approach, and showing its advantages. The second chapter presents a short description of the currently available workflow systems for Grid computing, and describes their architectural choices. In the same chapter there is also a more detailed description of the BPEL and BPMN and of the problems that arise when these two standard are adopted together. The third chapter presents our workflow system highlighting how we have approached the problem of translating a BPMN graph into a BPEL process. The fourth chapter gives a general description of the A-WARE project where our workflow system has been developed. In the fifth chapter we finally present our conclusion and future work plans.

2 Related Work and Background

Kepler [6] is a workflow system developed under several research projects. It is written in Java and based on Ptolemy II [6], a system originally created to study modeling, simulation, and design of concurrent systems. Kepler models workflow in MoMl, a markup language developed in Ptolemy that is based on the concept of actors. Each actor has input and output ports and wraps typical workflow objects like Web- and GridServices, Globus Grid Jobs, GridFTP, and many others. Therefore, Kepler can be classified as a system based on data flow (rather than sequence flow).

Taverna is an open source software tool for designing and executing bioinformatics workflows, developed in the myGrid project [7]. It uses Scufl as a runtime language for the execution of the workflow, and Freefluo as an execution engine. Scufl can be extended using Processor plug-ins that manage the interaction with different external service interfaces. Taverna is shipped with a set of processors that guarantee interaction with services based on WS-I standard. The designer is based on a directed graph that represents data flow, which uses a simple graphical notation developed by the Taverna team. The GUI can be extended creating new palettes and new widgets.

Several workflow systems have been developed in the enterprise world, some of them are:

- IBM WebSphere Process Server is part of the WebSphere framework, based on the BPEL language, and it can be used along with the WebSphere Business Modeler that provides designing capability in a notation that has been created specifically for BPEL processes
- Oracle BPEL Process Manager is part of the Oracle Fusion Middleware. It is based on the BPEL language and it has a designer based on another not standard notation.
- BEA AquaLogic BPM Suite is an integrated solution for creating, executing and optimizing business processes. It uses BPEL as its underlying runtime language and it has two designer applications: one for business users that allows drawing high level representations of the workflow, and another one for IT users that allows specifying low level details regarding service binding and data mapping. Both designer applications do not use standard notation.

Concluding in the enterprise world there is a convergence for BPEL based workflow execution systems, but concerning the notation, different providers offer different solutions that do not adhere to any standard. This approach causes problems to end users who want to create scientific workflows. In fact, they have to waste time learning a new notation every time they change the execution engine. On the other hand, in the Grid world both designers and engines use various technologies and approaches. In this case there is also the problem that a workflow created on a particular platform can not be executed on another platform because of the different engines.

We believe that a standard based approach especially in scientific communities with both designer and engine will assist the final user and above all will provide a better means to share work between scientists.

2.1 BPMN and BPEL

BPMN [8] is a standardized graphical notation to draw business processes, developed by the Object Management Group (OMG). Its primary goal is to provide a notation readable by all stakeholders, business analysts, scientific users, and IT specialists. It is based on a directed graph where the sequence of processes and the messages that flow between different process participants are coordinated in a related set of activities. The specification does not include any indication on how a graph should be saved into a file.

BPEL [9] is an orchestration language that is serialized in an XML format. It is developed by the OASIS group and has reached version 2.0 (commonly referred to as WS-BPEL). Essentially, it is an imperative programming language with specific constructs for Web Services interaction. BPEL does not specify how a process should be represented graphically.

Both standards have been developed independently and hence they do not take into account how to map a BPMN graph into a BPEL process. Currently there are two possible approaches, one restricts the possible graph that can be drawn in the designer, and the other one implements a complex algorithm that allows mapping a BPMN graph into a BPEL process. Section 3.2 provides a detailed discussion on this topic and it explains our approach.

3 Workflow Framework System

Our proposed solution is being developed within the A-WARE research project. A-WARE aims at creating a workflow system able to coordinate various instances of different Grid fabric middleware. Initially, UNICORE will be the reference platform [10], but support for other Grid systems is planned for the project. The other ambition pursued by the A-WARE project is to design and implement a pluggable workflow infrastructure flexible enough to support and host multiple workflow languages and engines. The additional challenge A-WARE is facing is to provide users with a fully Web-based interface for the whole workflow life-cycle management, from design to submission, from monitoring to result retrieval.

In the following paragraphs we outline some of the details of the A-WARE workflow system design and implementation. We identified three user roles for our system: the scientific/business logic user, the IT specialist and the end user.

- The scientific/business logic user defines new methods and processes in his specific domain of expertise within the company, designing workflow.
- IT specialists support business users with the IT infrastructure development. They participate in the process of design enriching workflow with all the technical details needed to make them executable against a specified engine.
- End or application users focus on accomplishing their business tasks. They parameterize and submit predefined business processes.

The user roles described above highlight the need to modularize the workflow design process into two stages: the first phase targeting the business expert role and the second phase involving the IT specialist. For the first release of the A-WARE system, the creation of a BPMN/BPEL workflow is actually performed in several steps, each one accomplished by one of the two previously mentioned user roles, the scientific/business user and the IT specialist, as depicted in the Fig. 1:

1. design: for the graphical representation of the workflow,
2. translation from BPMN to BPEL,
3. the grounding of BPEL for the creation of an executable workflow.

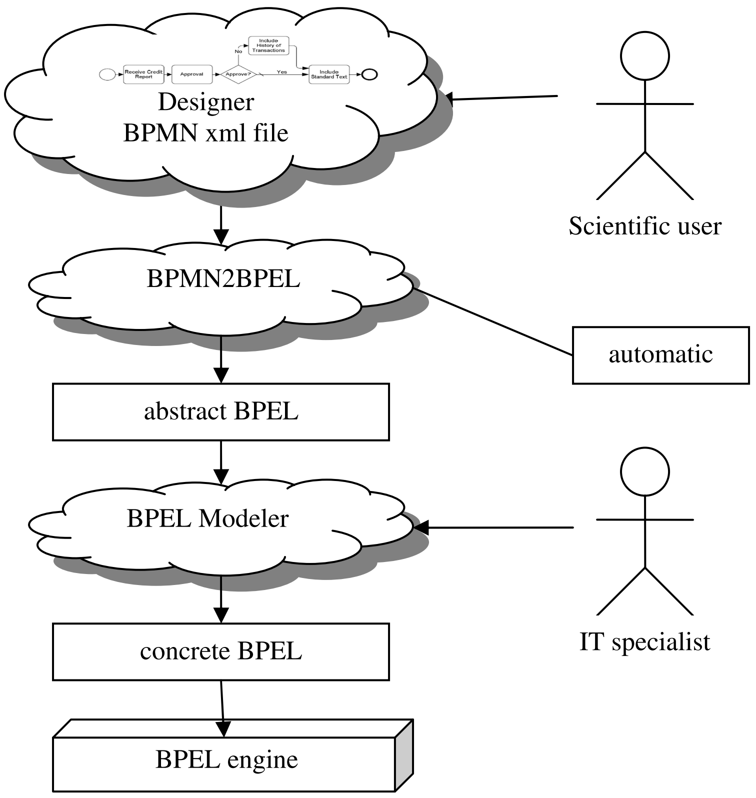


Fig. 1. Architecture of the Workflow Management System

In the next sections we describe each one of these three steps in detail.

3.1 Design of a New Workflow

The first design phase implies the construction of the graphical workflow model by placing nodes and edges within an editing environment offering drag and drop functionalities as well as other interactive, advanced editing modes. The component aimed to provide such functionalities is called the Workflow Designer Application or just Designer.

To implement the designer we have used a Java applet, which can be embedded in a Web page. For the implementation of the graphical part we relied on the Java Universal Network/Graph Framework [14], which provides a common and extensible library for modeling, analysis, and visualization of data that can be represented as a graph or network. Several BPMN elements have already been made available in the designer, as it is shown in Fig. 2, but others can be added if required. The designer can save the graph into an XML file.

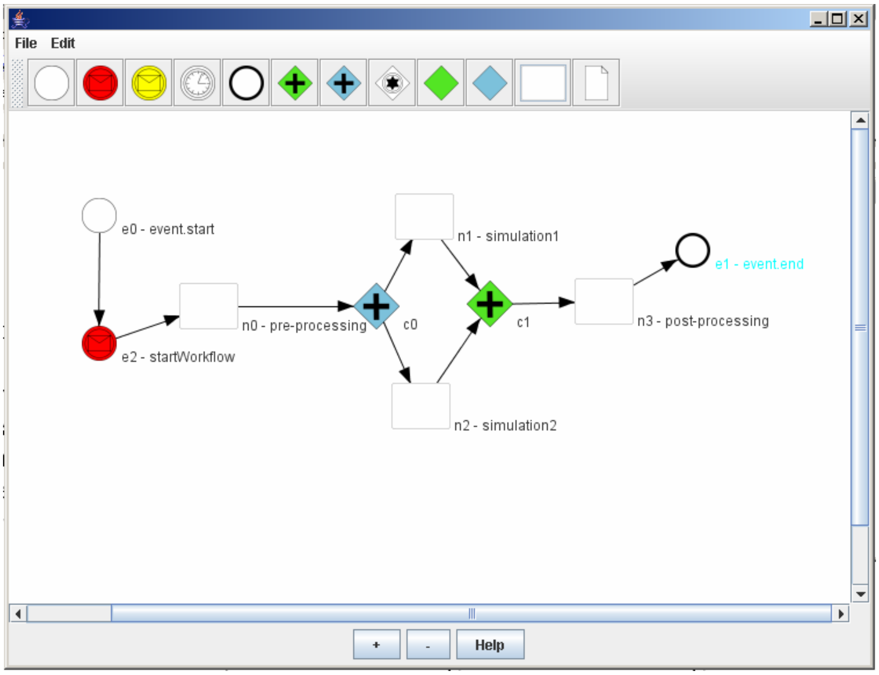


Fig. 2. Snapshot of the Workflow Designer

The designer does not let the user specify all the parameters required to execute the workflow, but permits to describe only the flow structures and a partial representation of the data flow. Firstly because [11] adding too much information directly into the graph can result in a clumsy and hardly understandable representation. Secondly, we wanted to separate low level IT details (like service binding and low level data mapping) from the scientific aspects of the simulation.

Finally, our designer is not bound to any particular workflow system. Its representation is completely independent from the underlying workflow engine.

3.2 Converting BPMN into a Workflow Language: BPMN2BPEL

The next step in the workflow creation process is the translation of the BPMN graph into a BPEL process description. As described in research literature [12], the two representations belong to different classes of languages. BPEL is mainly a block-structured language while BPMN is graph oriented. Mapping between these two different sets of languages is notoriously challenging.

S. White [11] sketches some guidelines to translate between BPMN and BPEL, but his solution is limited, because it restricts the possible topologies of the graph. To tackle this drawback C. Ouyang et al. [13] proposed an alternative algorithm which allows the automatic translation of any BPMN graph structure. This solution can create with most BPMN graphs a BPEL code that maintains the organization of the corresponding graph also called "well-structured" BPEL. On the other hand some particular graph topologies can be translated only by "event-action rules". Event-action rules are a block of BPEL code with additional control link to capture sequential dependencies between different activities. Obviously, this second translation produces code that does not directly reflect the original structure of the graph; hence it is more difficult to be read.

We believe that due to the complexity of the BPMN-to-BPEL mapping, enterprise applications have preferred not to use the BPMN notation in their designers. A designer application, adopting a different graphical representation can end up with an easier mapping between the graph and the BPEL process: the overall complexity of the system is lower but the graphical notation is domain specific or, worse, completely custom. On one side the A-WARE project brings the added value to be fully compliant with BPMN standard notation, on the other we had to cope with the complexity to translate BPMN to a BPEL detailed description.

During this phase the low-level information that would make the workflow executable by the engine is still missing, it needs to be provided by the IT expert user. For this reason the first transformation step produces an abstract BPEL that's compliant with the BPEL 2.0 specification [9] adopted by A-WARE.

From a high level point of view this module provides the translation from the BPMN notation into an executable workflow language. It actually decouples the notation from the underlying workflow language and engine. Moving in the direction to support other workflow languages, this library should be extended with pluggable translation modules.

3.3 Workflow Grounding: BPEL Modeler

As described above, the first transformation of the BPMN into a workflow language does not provide sufficient information to produce a working workflow ready for execution. To have a "concrete" (grounded or executable) workflow that can be run on an engine the IT user needs to provide the following missing information: the actual services incarnating the abstract workflow tasks and the actual data flowing through the workflow as input/output to/from the various tasks.

In the case of the BPEL processes this information will be:

- Service mapping, the service endpoints and binding protocols (e.g. port type, service name, port)
- Data mapping, the format of the exchanged messages by the various orchestrated services, e.g. XML selections with XPATH and manipulation with XSLT

Moving from an abstract BPEL to a concrete BPEL is commonly referred to as workflow grounding. In our system we have implemented a library that can manipulate and modify BPEL files. The A-WARE Web interface, currently based on the EnginFrame Grid portal framework [15], drives the IT user through a set of dynamic forms during the grounding process and triggers the BPEL modeler module on the base of the user's choices.

Both service binding and data mapping operations involve low-level infrastructural IT details that are usually quite distant from the scientific and business logic aspects of the workflow. This last point justifies the choice to have a different user interface for the two steps of the workflow creation process: design and workflow grounding.

4 The A-WARE Infrastructure

The orchestration engine for executing BPEL is based on the Apache ODE engine. This is hosted within the Java Business Integration (JBI) environment of the A-WARE Service Bus (ASB). JBI offers an SOA based framework for the integration of applications, data and processes, the normalization of otherwise incompatible protocols, and finally by providing a highly scalable and robust service environment for the management of processes, process instances, Grid interactions and all other supporting functionality. The Service-oriented computing environment of ASB enables a clean, pluggable platform where various connectivity and extensibility options are supported, various workflow execution strategies can co-exist, and through the semi-autonomous nature of the Grid component access to multiple Grid fabrics.

The ASB acts as a mediator between the Orchestrator (the BPEL engine) and the Orchestrated (the Grid resources). The common tasks necessary to interact with the Grid are exposed as higher-level functionality on the ASB using an intermediate proxy called the Grid component. They are expressed using higher-level formulations than typically found in Web Services based interfaces of the underlying Grid fabric. The ASB 'lifts' the applications defined as installed and available on the target systems to application-specific services on the bus. In this way users are able to express their requirements using terminology relevant to a specific targeted application. In addition, instead of the back-and-forth (request/response) nature of WSRF based Web Services for the Grid, the BPEL process orchestrates processes based on long-running, asynchronous but correlated messages running over the ASB. The Grid component acts in a semi-autonomous manner providing this functionality to its consumers on the bus. It manages the interactions with the un-predictable and un-reliable resources of the Grid fabric on behalf of its consumers.

From a monitoring perspective the ASB in conjunction with the ODE BPEL engine keeps track of each running instance. It provides monitoring capability tracked through capturing the messages sent from each running process instance to provide runtime 'picture' of the state of the process. It is also worth noting that this functionality is independent of orchestration strategy. However, it is possible for a particular orchestration engine to further embellish this monitoring information with engine-specific monitoring metadata.

5 Conclusions

In this paper we have seen how originally proprietary protocols moved towards service-oriented architecture to enable the interaction of services within heterogeneous infrastructures. Consequently, standards emerged that enabled the orchestration of services across boundaries of these infrastructures and allowed for the orchestration's description to be exchanged. BPEL is one such standard description. However, it lacks a specification of its graphical representation. As a result, different implementations of BPEL execution engines differ in their approach of graphically representing workflows. A standard notation for Workflow is available with the BPMN. Unfortunately, mapping between BPMN and BPEL and vice versa is a complex task. In this paper we have shown a possible approach based on an algorithm proposed by C. Ouyang et al. [13].

Users of a Grid system certainly don't care about the difficulty of mapping BPMN to BPEL; they want a simple solution that provides an easy to understand notation. The A-WARE software stack provides such a solution by separating responsibilities of Workflow design and deployment to experts with different roles and different Web interfaces.

In A-WARE, services are mostly provided as services on the ASB, which are orchestrated. Through the Grid component, they primarily access UNICORE 6 resources, although, through leveraging the Roctopus library, UNICORE 5 and other middleware are possible targets.

Future Developments. At the time of writing, the workflow design process is irreversible. Due to the complex mapping of BPMN to BPEL, currently it is not possible to translate an existing BPEL process into a BPMN graph. This hinders a seamless workflow development cycle for scientific and IT users. A solution where meta-data about the grounding process is stored along with workflows has been thought of, but still needs implementation.

Acknowledgments. This work has been supported by the European Union under the FP6 IST-05-034545 project. A special thanks goes to all the other members of the A-WARE project.

References

1. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration (February 2002), `http://www.globus.org/alliance/publications/papers/ogsa.pdf`
2. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture. In: W3C Working Group Note (February 2004)
3. Miyazaki, S., Sugawara, H., Ikeo, K., Gojobori, T., Tateno, Y.: DDBJ in the Stream of Various Biological Data. Nucleic Acids Research 32, 31–34 (2004)
4. Pillai, S., Silventoinen, V., Kallio, K., Senger, M., Sobhany, S., Tate, J., Valenkar, S., Golovin, A., Henrick, K., Rice, P., Stoehr, P., Lopez, R.: SOAP-based Services Provided by the European Bioinformatics Institute Nucleic Acids Research. 33, 25–28 (2005)
5. Fox, G.C., Gannon, D.: Special Issue: Workflow in Grid Systems: Editorials. Concurrency and Computation: Practice and Experience 18(10), 1009–1019 (2006)

6. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice & Experience 18(10), 1039–1065 (2005)
7. Oinn, T., Greenwood, M., Addis, M., Alpdemir, M.N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Stevens, R., Wipat, A., Wroe, C.: Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. Concurrency and Computation: Practice and Experience 18(10), 1067–1100 (2006)
8. Object Management Group: Business Process Modeling Notation (BPMN) 1.0: OMG Final Adopted Specification (February 2006)
9. OASIS: Web Service Business Process Execution Language Version 2.0. OASIS Standard (April 2007)
10. Streit, A., Erwin, D., Lippert, T., Mallmann, D., Menday, R., Rambadt, M., Riedel, M., Romberg, M., Schuller, B., Wieder, P.: UNICORE - From Project Results to Production Grids. In: Grid Computing: The New Frontiers of High Performance Processing. Advances in Parallel Computing, vol. 14, pp. 357–376. Elsevier, Amsterdam (2005)
11. White, S.: Using BPMN to Model a BPEL Process. BPTrends 3, 1–18 (2005)
12. Ouyang, C., Dumas, M., ter Hofstede, A.H.M., van der Aalst, V.M.P.: Pattern-Based Translation of BPMN Process Models to BPEL Web Services. International Journal of Web Services Research (to be published, 2007)
13. Ouyang, C., Dumas, M., ter Hofstede, A.H., van der Aalst, W.M.P.: From BPMN Process Models to BPEL Web Services. In: Proceedings of the IEEE international Conference on Web Services, vol. 00, pp. 285–292 (2007)
14. O'Madadhain, J., Fisher, D., White, S., Boey, Y.: JUNG: The Java Universal Network/Graph Framework., http://jung.sourceforge.net
15. EnginFrame Grid portal – NICE s.r.l.: `http://www.enginframe.com`

VHPC 2007: Workshop on Virtualization/Xen in High-Performance Cluster and Grid Computing
(Foreword)

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

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

As basic virtualization technologies mature, the main focus of research is now techniques for managing virtual machines in large-scale installations. This was reflected in this year's workshop, where five presentations were given on the management of virtualized HPC systems. It was good to see work integrating VMs into existing management systems such as SmartFrog and Quattor. In total ten papers were accepted for this year's workshop, with an acceptance rate of approximately 50%.

An invited talk by Greg Law of Solarflare described their implementation of high-performance I/O for guest VMs which was developed to support for their 10-Gbit Ethernet card. The model presented allows for direct, secure, low-latency access to network hardware from guest VMs. This addresses a significant limitation of Xen in an HPC environment, and would potentially allow many virtual cluster nodes to operate using the full potential of the underlying network hardware.

The Chairs would like to thank the Euro-Par organizers, the members of the Program Committee along with the speakers and attendees, whose interaction created a stimulating environment. Thanks also to Greg Law for accepting our invitation to speak at the workshop, and to the conference for their financial support which made this possible. XHPC/VHPC is planning to continue the successful co-location with Euro-Par in 2008.

November 2007

Michael Alexander
Stephen Childs

Virtualization Techniques in Network Emulation Systems

Roberto Canonico, Pasquale Di Gennaro, Vittorio Manetti, and Giorgio Ventre

Dipartimento di Informatica e Sistemistica
Università di Napoli Federico II
via Claudio 21, 80125 Napoli, Italy
{roberto.canonico,pasquale.digennaro,
vittorio.manetti,giorgio}@unina.it

Abstract. The continuous increase of computational power has made viable the implementation of more and more sophisticated virtualization techniques. The use of virtualization in cluster environments to build on-demand computing infrastructures is a recent trend with a great potential. Cluster-based network emulators are a specific class of cluster-based systems whose main purpose is to help researchers evaluate the effectiveness of new protocols and applications in realistic, synthetically generated network scenarios. Both large scale experimental testbeds (such as PlanetLab) and cluster-based network emulation systems (such as Emulab) use virtualization techniques at the basis of their resource management mechanisms to achieve isolation and concurrent experiments execution. In this paper, we compare different virtualization techniques already adopted in this kind of distributed systems and illustrate the peculiar virtualization requirements of a cluster-based network emulator. Furthermore, we show how Xen can be used to build a flexible and scalable network emulation system.

1 Introduction

In the last few years, network emulation has gained interest in the community of network reserchers, being considered an important technique to evaluate the effectiveness of new protocols and applications in heterogeneous, controllable and realistic network scenarios. In a network emulation experiment, simulated network elements interact, in real-time, with real network components and applications. Today's most complex network emulation systems are cluster-based. These systems are made of a large number of hardware components arranged in a common facility that can be remotely accessed by users through a web interface. Components include links, switches, routers, and PCs that interchangeably play the roles of end-systems, routers, or WAN emulators. An efficient use of the available hardware resources is one of the main goals, usually achieved by means of a combination of virtualization and space-sharing that aims at allowing simultaneous non-interfering experiments.

In a typical cluster-based network emulation system, users submit to the system an experiment request. An experiment request contains a "virtual" network description to be reproduced with the available cluster resources. A conservative resource allocation policy consists in mapping the emulated "virtual" nodes onto dedicated PCs and emulated links onto switched ethernet links. Nowadays, with increasing computational power made available at low-cost, it is possible to exploit virtualization techniques to map multiple "virtual" nodes on a single CPU. There are many good reasons for doing that. For example many applications need to be evaluated on large topologies, yet they are not resource hungry. Moreover, multiplexing provides a more efficient use of communication resources as the bandwidth of the emulated geographic links is usually much less than that available in the local interconnect used in a modern cluster [1]. These reasons motivate the use of small-scale clusters to emulate medium/large size topologies in a inexpensive manner [2],[3].

In the rest of this paper we will illustrate the importance of virtualization for network emulation. We compare different virtualization techniques already adopted in cluster-based network emulation systems. Furthermore, we show how Xen can be used to build a flexible and scalable network emulation system.

2 Cluster-Based Network Emulation Systems

Maybe the most complex cluster-based emulation system developed so far is Emulab [4]. Emulab is a free-for-use, Web-accessible, time- and space-shared, reconfigurable network testbed, providing integrated access to a wide range of experimental environments. The Emulab core consists of several hundred rack-mounted PCs, combined with secure, user-friendly web-based tools, and driven by ns-compatible scripts or a Java GUI, allowing remote configuration and control of machines. Even the OS of a cluster node can be fully and securely replaced with custom images by any experimenter.

In Emulab an enhanced version of FreeBSD jail has been used, which allows the creation of isolated environments (vnodes), characterized by independent namespaces. Each of these vnodes is accessible not only through the host node, but also remotely via ssh. This kind of virtualization technology used by Emulab is not resource hungry: this gives the opportunity to build a relatively large number of vnodes even on not very powerful machines. Anyway, this mechanism does not offer fully isolated execution environments, potentially creating some security issues. For this reason, in Emulab all vnodes running on a given physical host must belong to the same experiment. Even the network is not completely virtualized, since much of the network stack is shared between physical host and vnodes.

The Network Emulation Testbed project [5] provides a configurable network environment for the performance analysis of distributed applications and protocols, consisting of a 64 node PC cluster system running Linux connected by a flexible network infrastructure. The network infrastructure can be set up in arbitrary ways, emulating anything from Wide Area Networks (WANs) to highly dynamic Mobile AdHoc Networks (MANETs). The node PCs are connected by means of a Gigabit Ethernet switch on which an arbitrary number of VLANs can be configured. Through the use of VLANs, nodes can be connected with any possible virtual topology. Network traffic on emulated links is controlled by a special traffic shaper module, called NETshaper, implemented as a Linux kernel module. NETShaper provides a link-layer emulation that is completely transparent to upper layers. Parameters that can be emulated by NETShaper include fixed delay, variable delay, and frame loss.

NEPTUNE is a cluster-based emulation system developed at University of Napoli. Even though many design assumptions made for NEPTUNE were borrowed by Emulab, since from the early stages of design, NEPTUNE has assumed virtualization as a key technology for realizing complex networking scenario. The NEPTUNE project was started in 2004 at University of Napoli. The project main goal is to create a cluster-based network emulation system that could be used to assess either new networking technologies and protocols (e.g. to test new QoS Routing protocols and Traffic Engineering schemes in MPLS-based networks), as well as new distributed applications (e.g. multimedia peer-to-peer applications).

At the time of this writing, the NEPTUNE emulation system runs on a cluster of workstations consisting of 28 biprocessor nodes ProLiant DL380, each equipped with two Intel Pentium IV Xeon 2.8 GHz CPUs, 5 GB of PC-2100 RAM, one 100 Mbps Ethernet NIC, one Gigabit Ethernet NIC. Each node is equipped with a 34.6 GB SCSI disk. A 700GB centralized disk array is also available to the whole cluster. The cluster nodes are connected each other through a set of 100/1000 Ethernet switches.

One of the cluster nodes, the NeptuneManager, provides the fundamental services (like dhcp, dns, tftp, nfs, and so on) needed to properly configure at boot-time the physical cluster nodes and the virtual machines participating to the emulation experiments. A web-based system is used to manage and configure the whole system.

Setting up an emulation experiment in NEPTUNE consists primarily in defining a “virtual topology” made of emulated intermediate network nodes (routers) and end-system nodes (user terminals). A complex networked system can be reproduced by allocating multiple “virtual” network nodes (both routers and end systems) on each of the cluster physical nodes. A testbed mapping module (much like the one used in Emulab [6]) is responsible of mapping the “virtual” topology onto the cluster physical resources. Virtual network nodes are implemented in NEPTUNE as Xen virtual machines. The main advantage of the use of virtualization techniques to instantiate virtual network nodes is the significant reduction in equipment and management costs. Virtual machines allow the creation of customized execution environments, where customization consists in selecting the operating system, installed software packages and user access policies. Furthermore, virtual machines can be paused or shut down at any time, and later resumed, even at a different physical location (migration). Finally, virtual machines support fine-grained mechanism for resource usage control, allowing to define (and even change at run-time) precise limits to the the amount of usable RAM and disk space.

3 Virtualization Technologies

Virtualization is a widely used technique in which a software layer multiplexes lower-level resources among higher-level software programs and systems.

In a non virtualized system, a single OS controls all hardware platform resources. A virtualized system includes a new layer of software, the virtual machine monitor (VMM). A virtual machine monitor manages the creation, destruction and control of one or more virtual machines (VM) on a computer, and is responsible for controlling access to the resource of the real hardware, as well as multiplexing the execution of multiple VMs fairly. Virtual machines do not access the system's real resources directly, but through the VMM.

Some virtualization techniques support migration of virtual machines. In addition to facilitating hardware maintenance operations, VM migration can be triggered automatically by workload balancing or failure-prediction agents.

In the following we will present a few virtualization technologies that have been used in distributed experimental infrastructures to support multiple concurrent experiments.

3.1 FreeBSD Jails

The Emulab system supports multiple experiments running concurrently on the same physical node. This is implemented thanks to the use of the FreeBSD Jail mechanism. The FreeBSD Jail facility provides the ability to partition the operating system environment. Administrators can create several independent mini-systems called jails and provide access to the super-user account in each of these without losing control of the over-all environment. Each jail is a virtual environment running on the host machine with its own files, processes, user and superuser accounts. From within a jailed process, the environment is indistinguishable from a real system. A process in a partition is referred to as in jail. When a FreeBSD system is booted up after a fresh install, no processes will be in jail. When a process is placed in a jail, it and any descendents of the process created after the jail creation, will be in the same jail. A process may be in only one jail, and processes within the jail are prevented from delivering signals to processes outside the jail. The only way for a new process to enter the jail is by inheriting access to the jail from another process already in that jail. Processes may never leave the jail they created, or were created in. When a jail is created, it is bound to a particular file system root. Processes are unable to manipulate files that they cannot address, and as such the integrity and confidentiality of files outside of the jail file system root are protected. Security is simply guaranteed because the jail environment is separated from the rest of the system, in other words, since the jail is limited to a narrow scope, the effects of a misconfiguration or mistake does not jeopardize the rest of the system's integrity. Modifying the running kernel by direct access and loading modules is prohibited, just like modifying the network configuration and accessing raw, divert and routing sockets are prohibited. Thanks to the limited scope of a jail, it allows administrators to painlessly delegate several tasks which require superuser access without handing out complete control over the system. With jails it is possible to install different daemons in different jails and delegate their administration to other people by giving them access to the superuser account. It is safe because the jailed superuser has limited privileges and he can't escape the jail because he cannot get any information about the base system. Virtualization is valuable to service providers wishing to offer their users the ability to have custom configurations and yet keep the overall system easy to maintain.

3.2 Linux VServer

The Linux-VServer technology implements a soft partitioning concept based on Security Contexts which permits the creation of many independent Virtual Private Servers (VPS) running simultaneously on a single physical server. A VPS provides an almost identical operating environment as a conventional Linux server. All services can be started on such a VPS, without modification, or with only minimal modifications. The implementation of Security Contexts requires some modification to the plain Linux kernel. The purpose of a Context is to hide all processes outside of its scope, and prohibit any unwanted interaction between a process inside the context and a process belonging to another context. This separation requires the extension of some existing data structures in order for them to become aware of contexts and to differentiate between identical uids used in different virtual servers. It also requires the definition of a default context that is used when the host system is booted, and to work around the issues resulting from some false assumptions made by some user-space tools that the init process has to exist and to be running under id '1'.

The real drawback when VServer is used in a network emulation system, is that networking is based on isolation, not on virtualization. This prevents each virtual server from creating its own internal routing or firewalling setup. Furthermore, it is not possible to assign different MAC addresses to VPS.

3.3 OpenVZ

OpenVZ [7] is an another operating system-level virtualization technology built using GNU/Linux. It gives the ability to run multiple isolated system instances, called *Virtual Private Servers (VPS)* or *Virtual Environments (VE)*. It does not offer the same flexibility in the choice of the operating system, if compared to other solutions such as VMware and Xen, but in many usage scenarios it can be an interesting solution. Networking in OpenVZ is implemented through a virtual device (*venet*). Network emulation can also benefit of the use of Virtual Ethernet device (*veth*), that is an Ethernet-like device which can be instantiated inside a VE. A veth can be assigned a MAC address and used in a bridged configuration, emulating a sort of "virtual switch" inside the host, to which all virtual interfaces created in the VEs are connected. Each veth can be configured via dhcp at boot time, when the VE is started.

3.4 Xen

Xen [8] is a paravirtualization system developed by the University of Cambridge. Xen provides a virtual machine monitor for x86 processors that supports execution of multiple guest operating systems at the same time.

Today, a special Xen-compatible version of Linux, XenoLinux, is available. According to Xen researchers, 100 XenoLinux instances can be run simultaneously on a single Xen VMM with minimal performance degradation. Xen-compatible version of Windows XP and NetBSD are actively being developed at the time of this writing.

4 Virtualization for Node Multiplexing

Node multiplexing is the problem of emulating more than a network node on the same physical cluster node. This problem is inherently a problem of machine virtualization, as it has been described in the previous section. Hence, it can be solved with one of the many available virtualization technics. Aspect to be taken int account to select the proper one for a a cluster-based emulation system are efficiency, scalability, flexibility, isolation, and operating system customization.

In Emulab, an extended version of FreeBSD Jails is used. Jail allows the creation of different execution environments at the same time. The user can perform remote ssh to each of the execution environments. The Jails implementation is relatively lightweight, so it is possible to instantiate several execution environments on the same machine. Unfortunately, the Jails approach has also some drawbacks. In particular, since it does not rely on a virtual machine monitor, the degree of isolation among different execution environments is limited. To limits the negative effects of this problem, in Emulab, all the execution environments activated on the same physical machine must belong to the same emulation experiment. Furthermore, the communication resources are not virtualized at all: the whole protocol stack is shared among the various execution environments. This, incidentally, make it impossible for Jail to support communication with guaranteed Quality of Service. Finally, the Jail mechanism is currently only available in the FreeBSD kernel, and its porting to other platforms does not appear straightforward.

The node multiplexing technique we chose for NEPTUNE is Xen, due to its many advantages, and in particular because Xen is:

- highly scalable;
- potentially supports different kinds of Operating Systems;
- provides good isolation among different virtual machines running concurrently;
- supports virtual machine migration, allowing dynamic re-allocation of experiments on the cluster nodes;

and also because Xen implements different optimization techniques in the communication mechanisms, allowing good communication performance among virtual machines implemented within the same physical node.

5 Virtualization for Link Multiplexing

The nodes of a cluster are connected by means of one or more switched Ethernet LANs. Each cluster node may be equipped with one or more (Giga or Fast) Ethernet NIC. These NICs, in turn, may be connected to the same switch or to different switches. In theory, it would be possible to connect the cluster nodes in pairs, by means of crossed Ethernet cables, so to physically reproduce the desired topology. However, such a solution is not viable, for at least two reasons. Firstly, because changing the network topology would be extremely impractical, time-consuming and error-prone. Secondly, because this would make it impossible to emulate network topologies with a number of links greater than half of the number of Ethernet NICs. Hence, practical solutions require to emulate multiple point-to-point connections on top of one or more shared Ethernet LANs. This is usually performed by means of Virtual LANs (VLANs) [1], [5]. Such a solution is implemented by properly configuring the Ethernet switches and does not require any configuration and processing in the cluster nodes. This makes, however, the system configuration software extremely dependent on the characteristics of the network switches. For the above reasons, we decided not to use VLANs in NEPTUNE and we adopted two network device independent solutions for link multiplexing:

- IP-aliasing and destination MAC address filtering
- Virtual NICs

The first technique is our choice when the emulated node has been mapped directly onto a physical node, i.e. there is no node multiplexing. The other solution consists in activating a virtual NIC and binding it to one virtual interface of a virtual machine. Traffic shaping is obtained by means of queuing disciplines directly attached to the virtual interfaces. Main advantages of this technique is a more clean management of the networking during the experiment creation and the possibility to use classless shapers in addition to classful ones, which are required to emulate characteristics of emulated links when ip aliasing is used.

6 Virtualization Techniques for Link Multiplexing Compared

In this section we show a comparison of two network emulation solutions, implemented by means of two different virtualization techniques both supporting the creation of a virtual interface inside a VM, namely OpenVZ and Xen. OpenVZ is an example of an operating system-level server virtualization solution, while Xen is an example of an hypervisor based on the paravirtualization concept.

Our emulated network consists of two end systems interconnected by means of a couple of intermediate IP routers. We compare two different implementations of this emulated network: one in which routers are implemented in Xen VMs, and another in which routers are implemented in OpenVZ VMs. In both cases we allocated the two VMs in two different Linux boxes. We also implemented a reference scenario, in which the intermediate routers are plain Linux boxes. The three scenarios are depicted in 1.

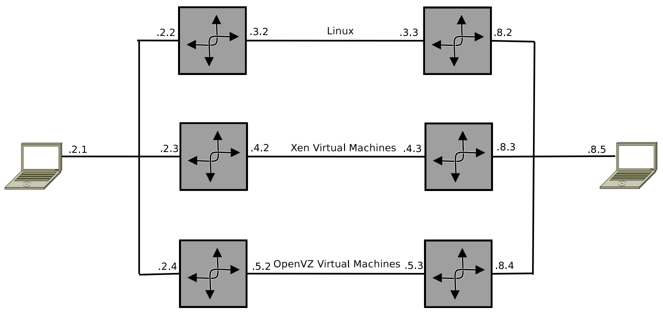


Fig. 1. Our experimental testbed

Physical machines have an identical hardware configuration: each of them is a SMP system with two Intel Xeon running at 2.8Ghz and is equipped with 5GB of RAM. For our tests we have used only one Tigon 3 gigabit ethernet interface on each machine. Virtualization of the network is obtained by means of IP-aliasing. In Xen we have created for each virtual router two virtual NICs. Xen networking has been configured in the bridge configuration: in dom0 a bridge has been created to which all virtual interfaces are connected. OpenVZ offers two networking implementations: *virtual ethernet* and *virtual network*. We have chosen the first one, as it gives the opportunity to assign MAC addresses to virtual interfaces.

Constant bit rate traffic has been generated between end hosts by use of D-ITG traffic generator [9], varying packet size and inter-departure time (IDT). In figure 2 we show packets dropped at increasing inter-departure rates; one Kbyte packets were used. Results demonstrate that for trasmission rates not exceeding 30000 packets per second, we almost don't have losses. Reducing inter-departure times, causes an increasing drop rate. Drop rates are always higher in Xen as compared to OpenVZ. Anyway, in both virtualization solutions, losses reduce the sustainable effective throughput in such configurations.

The mean jitter measured at the receiver node (estimated on 10 milliseconds intervals) has been plotted in figures 3, 4, 5 for the three scenarios . Since the experiment has been conducted in absence of any other interfering traffic, jitter is entirely due to the variable packet processing time in the intermediate routers. This experiment has been conducted by generating CBR streams made of 1KB packets transmitted at an increasing rate. Each stream has been generated continuously for 30 seconds. The comparison of results shows that OpenVZ routers introduce, on average, 50% less jitter than Xen. Similar results have been noticed varying packets generation rates.

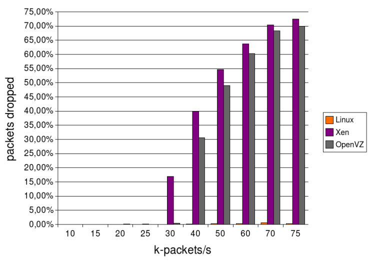


Fig. 2. Packets dropped vs. tx rate

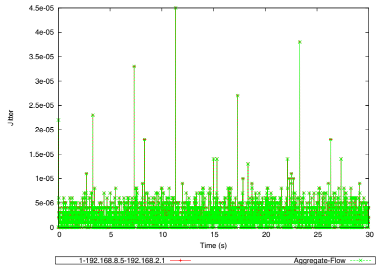


Fig. 3. jitter introduced by Linux routers

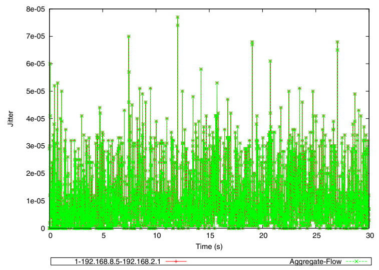


Fig. 4. jitter introduced by Xen routers

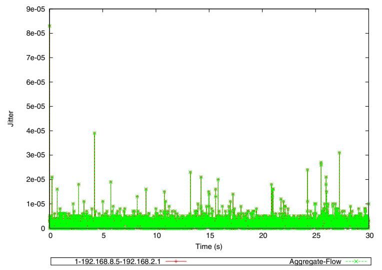


Fig. 5. jitter introduced by OpenVZ routers

7 Conclusions

Virtualization is maybe the most important tool to design effective network emulation systems. Virtualization of resources, in fact, offers two fundamental benefits: first, it allows to allocate more than one network node per single physical node; secondly, it is an instrument to allocate the available computational resources of a cluster among different users and different experiments, in an on-demand manner. When it comes to network emulation, it is not only important to virtualize CPUs, but also communication resources, e.g. network interfaces and their available bandwidth. For this reason, the designers of a cluster-based network emulation system need to take this specific requirement into account when they select the virtualization technique to be used at the foundation of their system. Our tests show that most of today's virtualization techniques have some problems when they have to deal with huge amount of network traffic. Our preliminary results show that OpenVZ performs better than Xen. Nonetheless, Xen is largely more easy to use and more flexible, and this is a great advantage for cluster-based systems of several tens or hundreds of machines. Furthermore, the fault-isolation capabilities of paravirtualization systems also play in favor of Xen. For all the above reasons, we decided to base the NEPTUNE emulation system on Xen, being confident that most of today's limitations will be overcome in future releases.

Acknowledgements

This work has been partially funded by the FP6 IST-2005-034819 project *OneLab: an open networking laboratory supporting communication network research across heterogeneous environments*. Pasquale Di Gennaro has been funded from the Italian Ministry of University and Research (MIUR), in the form of a grant in the framework of the S.Co.P.E. Project.

References

1. Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., Lepreau, J.: Feedback-directed virtualization techniques for scalable network experimentation. Technical report, University of Utah, Flux Group Technical Note 2004-02 (May 2004)
2. Guruprasad, S., Ricci, R., Lepreau, J.: Integrated network experimentation using simulation and emulation. In: Proceedings of TridentCom, IEEE, Los Alamitos (2005)
3. Maier, S., Herrscher, D., Rothermel, K.: On Node Virtualization for Scalable Network Emulation. In: Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005), Philadelphia, PA, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Simulation Councils, Inc., July 24–28, 2005, pp. 917–928 (2005)
4. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: Proc. of the Fifth Symposium on Operating Systems Design and Implementation, pp. 255–270. USENIX Association, Boston (2002)
5. Herrscher, D., Leonhardi, A., Rothermel, K.: On node virtualization for scalable network emulation. In: Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005) (July 2005)
6. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. SIGCOMM Comput. Commun. Rev. 33(2), 65–81 (2003)
7. OpenVZ server virtualization open-source project, `http://openvz.org`
8. 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 2003, pp. 164–177. ACM Press, New York (2003)
9. Avallone, S., Emma, D., Pescapè, A., Ventre, G.: Performance evaluation of an open distributed platform for realistic traffic generation. Performance Evaluation (Elsevier) 60(1-4), 359–392 (2005)

SOA Based Control Plane for Virtual Clusters

Paolo Anedda[1,2], Simone Manca[1], Massimo Gaggero[1], and Gianluigi Zanetti[1]

[1] CRS4, Center for Advanced Studies,
Research and Development in Sardinia,
Parco Scientifico e Tecnologico,
POLARIS, Edificio 1, 09010 Pula (Ca), Italy
[2] DIEE (Department of Electric and Electronic Engineering),
University of Cagliari,
Italy
{paolo.anedda,simone.manca,
massimo.gaggero,gianluigi.zanetti}@crs4.it

Abstract. Virtualization is an essential enabling technology for the construction and control of computing facilities that can dynamically adapt available physical resources to transient tasks such as the temporary creation of a virtual computing center tailored to the needs of a virtual organization. In this paper we will describe our strategy for the creation of virtual computer clusters based on standard SOA and hosts virtualization technologies and we will report on our ongoing work on the application of the latter to the deployment and management of a research cluster with 140 dual core cpu. Our deployment mechanism, as well as the system management, is delegated to a control plane based on workflows of coordinated web services. The control plane is based on two logically independent modules, the first is responsible of the physical resources and the deployment on the hardware of virtual Xen hypervisor images, while the second manages operations on virtual clusters such as their creation, startup and control. Low level operations – e.g., the control of a running image on a given computational host – are directly provided by atomic web services, in this specific case a WSRF service running in the dom0 of each participating physical Xen host, while all logic above that level is implemented as BPEL scripts.

1 Introduction

Recent advances in virtualization technologies and the exponential growth in network bandwidth are introducing new dimensions for the computational facilities configuration space and opening interesting and effective new ways to deliver High Performance Computing (HPC) resources to applications. In this paper we will report on our ongoing work on experimenting with these technologies in the context of Cybersar, a new computational infrastructure for research being currently built in Italy.

1.1 Virtual Computing Facilities

Virtual machines, such as VMWare [1], Xen [2], and KVM [3], bring to HPC two main gifts. On one hand, they enable the specialization of operating systems to particular tasks with hardware resources safely and transparently multiplexed by the vm hypervisor [4]. On the other hand, virtual machines allow the decoupling of the physical computational infrastructure management from the management of the user-visible virtual computing facility [5,6], making the former essentially homogeneous and agnostic with respect to the specific applicative context, while putting the latter in the hands of the virtual organization managing the virtual resource [7,8]. This separation removes many of the difficulties that oppose the effective sharing of computational resource between organizations that are not adhering to very strict common standards for their software stacks.

Virtualization guarantees a high level of flexibility in the dynamic configuration and use of "atomic" computational resource, which can then be organized and orchestrated to provide virtual clusters that, differently from standard, general purpose, HPC computing facilities, can be tailored to satisfy the needs of specific virtual organizations [9,8].

Virtual clusters can encompass computational resources that are distributed on different, potentially remote sites, current trends on ever increasing available network bandwidth [10] make it possible – at least when the structure of the applications run allows to trade latency for pipeline depth [11] – to tightly couple geographically distant computational resources in a single computational facility that appears as a coherent entity to its users. Such a constructions, however, requires a much closer coordination between computational and network resources with dynamic network configuration mechanisms, similar to the ones described in [12], being activate in parallel to the virtual cluster construction.

1.2 Cybersar

Cybersar is a high performance computing initiative recently (March 2006) funded by the Italian Ministry of Research that has as its goal the development of a Cyberinfrastructure for research in Sardinia based on high speed networks interconnecting all main scientific computational facilities and research communities of the island. Cybersar network core is on dark fibers and it provides an optical network backbone that can support application driven dynamic creation of point to point multi-lambda, currently 1GbE to be soon upgraded to 10GbE per lambda, optical circuits. The optical backbone is linked to the DWDM (Dense Wavelength Division Multiplexing) Regional network (2.5Gbit/s per lambda) and via the latter to the Janna submarine optical cables connecting Sardinia to the Italian Mainland and to Sicily. The main Cybersar computational resources are hosted by physically separated (20 to 70 km apart), computing facilities based at CRS4, the University of Cagliari, Cagliari Astronomic Observatory and the University of Sassari. The total computing power directly available to Cybersar is of about 1200 AMD Opteron class cores with about 2.5TB of RAM and more than 200TB of available disk space. Besides being a source of computing power for the local research community, and to contribute to the National computing infrastructure, Cybersar main goal is to be an experimental platform for research on new approaches to high performance computing.

2 Generalized Computing Control Plane

The dynamic construction of specialized virtual clusters will be the main mechanism that will be used within Cybersar to support large scale applications spanning the resources of multiple sites. This will allow, as it has been discussed above, to insulate the system managers of the partecipating sites from the specifics of operating system and middleware configuration while, at the same time, supporting reasonably fast procedure for application instantiation, freeze and tear down. On the other hand, as we will discuss below, the control plane machinery that we are setting up has, at its most basic layer, general mechanisms to deploy complete systems starting from the bare hardware. Thus, if the overhead to be payed to the virtualization layer is too high (e.g., when one needs to drive specialized hardware such as Infiniband PCI-Xpress boards), we mantain the option of mapping specific applications, similarly to what it is done in Grid5000 [13], to dynamically build real clusters. The logic behind the creation and the management of a virtual clusters, that we can represent as a set of information work-flows of cooperating components, is demanded to the control plane. Making a parallel with the networking field, the latter can be seen as the abstraction layer to which all the logic for the setup of the virtual computing center is entrusted. It is composed by all the programs and system architectures that are required to successfully deploy a new virtual cluster following a cluster blue-print. We consider the control plane as divided in two main components. The first is responsible for the physical allocation of the computational nodes and network resources, while the second is in charge of creating and administering virtual resources. For the practical implementation of the system we choose to adhere to SOA [14] principles and, in particular, to the W3C style of implementation. The entire system is basically build upon web services and is implemented using standard languages and protocols. Following the work of [15], we use Xen as a virtualization layer and web services to control virtual hosts. Since the creation of virtual clusters can be seen as the results of a process of web services calls, all control logic, above low level operations e.g., the control of a running image on a given computational host is implemented as work-flows.

2.1 Physical Resources Management

To be able to deploy complete systems starting from the bare hardware, we have developed a new deployment system, called HaDeS . It meets two main requirements. The first is to be able to communicate with an orchestrator, using standard SOA mechanisms, to properly control the critical steps of the deployment process such as creation of a specific disk partitioning, filesystems, and the population of partitions with the operating systems. The second is to be agnostic with respect to the operating system to be deployed. This is a major requisite for two reasons: we want to be able to support a large number of OS types with different installation methods; we want to be able to deploy a pre-assembled image of an operating system that can't be installed with traditional methods. These are requirements that systems such as RedHat Kickstart, or Suse Yast2, or even Debian FAE cannot meet. While other deployment system, such as SystemImager, are not designed to negotiate with an orchestration system. HaDeS is similar in spirit to kadeploy [16], albeit with a different implementation, but, differently from the latter, has been thought from the beginning to be integrated in a SOA architecture.

2.2 Virtual Resources Management

As reported in [6], there could be various levels or aspects of virtualization. In our work, when we talk about virtual resources, we refer to the possibilities offered by the so called virtualization software. Following this assumption, we define a virtual host as the abstraction of a real computing facility from which it inherits all the functionalities, running on top of a virtualization software. Going forward in this reasoning, a virtual cluster is composed by a set of cooperating virtual hosts connected by a virtual network.

Virtual Hosts Management. The lifecycle of a virtual host can be represented as a state diagram characterized by six states.

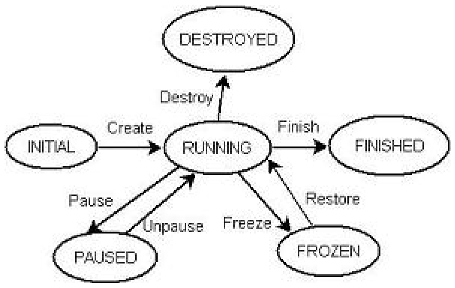


Fig. 1. The lifecycle of a virtual host

As depicted in the figure above, the lifecycle of a virtual host starts from the Initial state in which the virtual host is only a representation of a potential running host. When it is in this state, we call it a "Virtual Host Image" (VHI). A VHI is characterized by an operative system, a configuration and some representation of the initial conditions. We can represent it with a triple in the space of the virtual hosts:

$$(OS_i(t_0), C_i(t_0), Ic_i) \tag{1}$$

where we use OS_i, C_i and Ic_i to indicate, respectively, the operating system, its configuration and the initial state (initial conditions) of the virtual host.

When the resources controller decides to create a new virtual host, it turns a VHI into a running virtual host (VH) by instantiating it on a physical host. This transition is fired by a "create" event. After his creation, a virtual host evolves into the running state. A VH is represented by the same triple of the starting VHI at the time t, plus the reference to the VHI from which it was created:

$$(OS_i(t), C_i(t), Ic_i, Ref(VHI_i)) \tag{2}$$

From this state, a VH can be destroyed, paused or frozen. When it is paused all the VH's operations are suspended, while when it is frozen, the controller serialize it into a file and release all the resources associated with that particular VH. From this state it can be instantiated again in the same machine or, after a migration procedure, in another one.

The very basic building block of the system is the Virtual Host Controller (VHC). The VHC is the component responsible to instantiate and control the VHs inside a real host. The process of creation of each VH is controlled by the Virtual Host Factory (VHF). The VHF asks the appropriate VHC to instantiate a new VH using a specific VHI. The VHF knows exactly where the VHCs are and how to communicate with them. It can choose a particular VHC using a special algorithm for the resources' optimization, or on a random basis.

All the VHIs are maintained by the Virtual Host Images Repository (VHIR), that stores all information regarding a particular VHI and make them available to all the others components.

The Virtual Host Manager (VHM) is responsible for the orchestration of all the components' activities.

The creation of a Virtual Cluster (VC) is managed by the Virtual Cluster Manager (VCM). All the operations to instantiate a new VC are performed by the Virtual Cluster Factory, which is responsible to dialog with the VHM for the VHs creation. Its responsibilities include also the managing of all information about a particular VC that is running.

2.3 Virtual Resources Implementation

Virtual Host Controller. As we mentioned before, the very basic building block of the entire architecture is the VHC. This is the component that physically controls the creation and the management of the virtual machines running onto the physical hosts, through a common interface that wraps the command line calls to the Xend hypervisor control program. Each physical node is equipped with a WSRF (Web Services Resources Framework, [17]) container which contains an implementation of the interface. This allows to export, through a standard web interface, the main functionality of Xen to the rest of the system.

Virtual Host Images repository. The VHIR is the component responsible to manage the information regarding the "potential" virtual hosts. It is the repository where all the information about all the VHIs are stored.

A VHI is the description of the components of a VH in his initial state. It is composed by an image of an operative system, an initial configuration and the initial conditions.

The initial configuration is the set of information regarding the network, the device drivers and the storage resources.

The initial conditions are a representation of the state of the VH at the moment of his first running. With this, we mean a representation of the virtual memory and the filesystem associated with it. That's because a running VH can be frozen and his image, with all the virtual memory and the filesystem, can be stored to be used as a new image for the instantiation of a new VH.

When the VHIR is asked to give the information about a specific VHI, it knows how to retrieve the physical image of the operative system and creates a reference to it. It also has a description about the configuration of the network and the storage resources.

These information are usefull to create the connections among the VHs and are set at the moment of the creation of the VHI.

Virtual Host Factory. The VHF is the component responsible for the instantiation of a new VH. It receives the request, that indicates the type of the node to create and the resources associated to it, from the VHM. Then the VHF asks the VHIR for that particular image and evaluates the answer. If the request can be fulfilled, a VHI reference is sent from the VHIR to the VHF. At this point, because the VHF has a database with all the available VHCs and knows the state of every node, it asks a particular VHC to create a new VH passing the VHI reference to it. At this point the VHC creates a new VH and, if all goes well, starts it and returns the VH's reference to the VHF. The VHF updates his internal database with all the information regarding the new node and returns the new VH's pointer to the VHM.

Virtual Cluster Factory. The VCF is the component that is responsible for the creation of virtual clusters. When it receives a request from the VCM, it dialogs with the VHM for the creation of the VHs. For every VH specified in the VCM's request, it talks to the VHM.

It maintains an internal database of all the VCs available and of all the VHs belonging to each VC.

Work-flow control. The logic behind the creation and the management of virtual machines and its related network resources, can be expressed in terms of work-flows of business processes. A work-flow can be represented using a standard language. According to our philosophy to follow the SOA requirements, we choose the BPEL (Business Process Execution Language, [18]) language.

BPEL defines business processes that interact with external entities through Web Service operations defined using WSDL 1.1 (Web Service Definition Language, [19]). It defines business processes using an XML based language but do not define a graphical representation of processes or provide any particular design methodology for processes. Each business process has inputs, method and outputs and so, also the workflow exports a web-service interface. Workflows can be nested, so it is possible to call a workflow from inside another one. This allows us to define very complex execution processes.

Once is defined, a workflow is executed by a BPEL engine which is responsible for the instantiation of all the components defined and to coordinate their execution. The execution of a specific workflow is started by the invocation of a method defined by his interface.

The creation of VH can be represented as a sequence of coordinated actions performed by the components defined in the section above. Since the VHM is responsible for these operations, we decided to implement it as a workfow.

The process of creation of a new VH starts with the invocation of the create VH method of the VHM. This envent triggers the execution of the process depicted in figure 2. The VHM calls the VF with a request for a new VH. The VH asks the VHIR for the appropriate VI, according to the inputs from the VHM. The VHIR then returns a reference to the VI that meets the request's specifications. At this point, the VHF calls a VHC for the VH creation, passing the VI reference to it. The choice of a particular

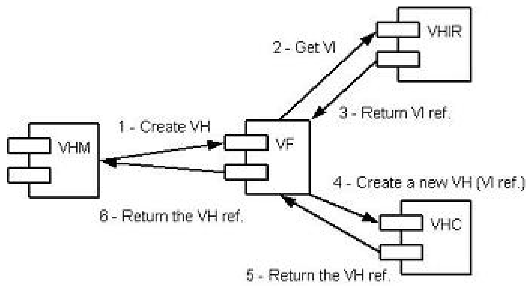


Fig. 2. The VH creation process

VHC, is made according to a specific workload policy. Once a new VH is created, the VHC returns a VH reference to the VF which sends it back to the VHM.

As we did for the VHM, we also implemented the VCM as a workflow. In this case the process of creation of a new VC, is represented by the VCM. When the VCM receives the request for a new VC, it invokes the create VC method on the VCF. The VCF then calls the VHM for the creation of a new VH as many times as the number of nodes defined in the VC creation's request. Once all the VHs are created, the VCF stores all the information regarding the VHs and returns to the VCM a reference to the new VC created.

Virtual Resources Definition. Following the approach of [15] and [9], to describe a VC we use an XML schema that defines all the details and the attributes of the corresponding computational resource associated to it. The XML describes the virtual cluster as a whole, in terms of general properties common to all the nodes; it also specifies the details of each node in terms of their parameters. So, a virtual cluster is defined by declaring a list of nodes. For each node, the name and the node type are specified. Each node can also contain the details of his implementation or a reference to a specific node configuration.

The schema definition has been splitted into two separate files. The former contains the description of the whole cluster while the latter the description of a single node. This was done to separate the level of details between the cluster definition and the nodes description. During the creation of a new cluster, the control plane, which is responsible for the new cluster deployment, processes the XML file and, for each node, it sends the corresponding XML fragment to a virtual host controller. The virtual host controller doesn't need to know about the whole cluster details; he only has to deal with the parameters necessary for the creation of a new virtual node.

A virtual node configuration contains the details of the disks partition, the network settings and also the boot parameters like the number of virtual processors or the amount of memory associated with the new host.

3 Preliminary Testbed

Following the approach described above, we develop the whole system using a preliminary experimental environment composed by three servers connected to the same ethernet switch. Each server was equipped with two 2.2 GHz AMD Opteron CPUs, 2GB main memory, and two 200GB hard drives.

One server was used for the implementation of the control plane. We choose the ActiveBPEL server[20] as workflows manager. ActiveBPEL is an open source implementation of a BPEL engine, that follows the J2EE ([21]) specifications, entirely written in Java. We deployed it onto the Tomcat container ([22]).

The other two servers were equipped with the Xen software. The VHs are created through the call to a local web-service running inside a WSRF container. The implementation of the web services was written in Python, using the pyGridWare toolkit [23].

To create all the scripts for the activation of the container and his associated services, we used a template mechanism ([24]) entirely written in python that, starting from a definition of the methods written in XML, is able to automatically generate all the stub classes and the code that are necessary for the execution of the service's methods.

4 Target Testbed

The actual target testbed for the control plane software is currently being delivered at CRS4 site, and it is composed by 72 IBM System x3455 machines. Each machine contains 2 dual-core AMD Opteron processors of the latest "revision F" series (2218) operating at 2.6 GHz, with 64 bit extension and 1 MB of L2 cache per core. The amount of RAM is of 8 GB per machine, while the storage consists of two 200 GB SATA disks, managed by the Broadcom SATA/Raid controller. Two Broadcom 5704 Gigabit Ethernet allow each host to connect to the LAN while their IPMI integrated controller allows remote monitoring and remote management.

Twenty-four machines of the 72 are connected, through an Infiniband Interface, to a Cisco Infiniband Fabric Switch. All nodes have two free PCI Express slots that can be used for further addition of new peripherals.

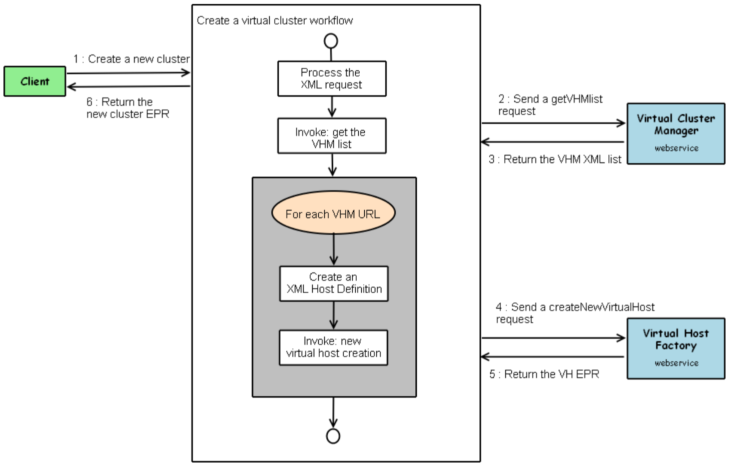

Create a virtual cluster workflow
Client
1 : Create a new cluster
6 : Return the new cluster EPR
Process the XML request
Invoke: get the VHM list
For each VHM URL
Create an XML Host Definition
Invoke: new virtual host creation
2 : Send a getVHMlist request
3 : Return the VHM XML list
Virtual Cluster Manager
webservice
4 : Send a createNewVirtualHost request
5 : Return the VH EPR
Virtual Host Factory
webservice

Fig. 3. The workflow of a new virtual cluster

5 Conclusions

We have described our strategy for the creation of virtual computer clusters based on standard SOA and hosts virtualization technologies and our ongoing work. Our deployment mechanism, as well as the system management, is delegated to a control plane based on workflows of coordinated web services. The control plane is based on two logically independent modules, the first is responsible of the physical resources and the deployment on the hardware of virtual Xen hypervisor images, while the second manages operations on virtual clusters such as their creation, startup and control. Low level operations – e.g., the control of a running image on a given computational host – are directly provided by atomic web services, in this specific case a WSRF service running in the dom0 of each participating physical Xen host, while all logic above that level is implemented as BPEL scripts.

Acknowledgments. This research is partially supported by the Italian Ministry of University and Research, project PON-Cybersar 2006.

References

1. Warren, S.S.: The VMWare Workstation 5 Handbook. Charles River Media, Hingham, MA, USA (2005)
2. Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., Neugebauer, R.: Xen and the art of virtualization. In: Proceedings of the ACM Symposium on Operating Systems Principles (October 2003)
3. Qumranet: Kernel based virtual machine., `http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine`
4. Mergen, M.F., Uhlig, V., Krieger, O., Xenidis, J.: Virtualization for high-performance computing. SIGOPS Oper. Syst. Rev. 40(2), 8–11 (2006)
5. Keahey, K., Foster, I.T., 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)
6. Adabala, S., Chadha, V., Chawla, P., Figueiredo, R., Fortes, J., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L., Zhu, X.: From virtualized resources to virtual computing grids: The in-vigo system. Future Gener. Comput. Syst. 21(6), 896–909 (2005)
7. Figueiredo, R.J., Dinda, P.A., Fortes, J.A.B.: A case for grid computing on virtual machines. In: ICDCS 2003: Proceedings of the 23rd International Conference on Distributed Computing Systems, Washington, DC, USA, p. 550. IEEE Computer Society Press, Los Alamitos (2003)
8. Ramakrishnan, L., Irwin, D., Grit, L., Yumerefendi, A., Iamnitchi, A., Chase, J.: Grid allocation and reservation—toward a doctrine of containment: grid hosting with adaptive resource control. In: Löwe, W., Südholt, M. (eds.) SC 2006. LNCS, vol. 4089, p. 101. Springer, Heidelberg (2006)
9. Foster, I.T., Freeman, T., Keahey, K., Scheftner, D., Sotomayer, B., Zhang, X.: Virtual clusters for grid communities. In: CCGRID, pp. 513–520. IEEE Computer Society, Los Alamitos (2006)
10. Foster, I., Grossman, R.L.: Data integration in a bandwidth-rich world. Commun. ACM 46(11), 50–57 (2003)

11. Smarr, L., Chien, A.A., DeFanti, T.A., Leigh, J., Papadopoulos, P.M.: The optIPuter. Commun. ACM 46(11), 58–67 (2003)
12. Lehman, T., Sobieski, J., Jabbari, B.: Dragon: A framework for service provisioning in heterogenous grid networks. IEEE Communications Magazine 44(3) (2006)
13. Cappello, F., Caron, E., Dayde, M., Desprez, F., Jeannot, E., Jegou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Richard, O.: Grid 5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In: Grid 2005 Workshop, Seattle, USA, November 13-14, 2005, IEEE/ACM (2005)
14. Hégaret, P.L.: Web services and soa., http://www.w3.org/2003/Talks/1211-xml2003-wssoa
15. Zhang, X., Keahey, K., Foster, I., Freeman, T.: Virtual cluster workspaces for grid applications. In: Cluster Computing and the Grid, 2006. CCGRID 2006. Sixth IEEE International Symposium (2006)
16. Kadeploy team: Kadeploy, http://kadeploy.imag.fr/
17. Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D., Tuecke, S.: Modeling and managing state in distributed systems: The role of OGSI and WSRF. Proceedings of the IEEE 93, 604–612 (2005)
18. BPELSource: Bpelsource, http://www.bpelsource.com/
19. World Wide Web Consortium: Web services description language (wsdl) 1.1., http://www.w3.org/TR/wsdl
20. ActiveBPEL: Activebpel., http://www.activebpel.org/
21. Sun: Java platform, enterprise edition., http://java.sun.com/javaee/
22. Apache: Apache tomcat., http://tomcat.apache.org/
23. Boverhof, J.: pygridware: Python web services resource framework
24. Rudd, T.: Cheetah - the python-powered template engine

Grid Virtual Laboratory Architecture

Eduardo Grosclaude, Francisco López Luro, and Mario Leandro Bertogna

Department of Computer Science
Universidad Nacional del Comahue
C.P 8300. Buenos Aires 1400. Neuquén. Argentina
{oso,flopez,mlbertog}@uncoma.edu.ar

Abstract. This work describes an approach to managing networked virtual and physical resources under a Grid-based Virtual Organization, viewed as elementary components of Virtual Remote Laboratories. While keeping overhead at local organizations at a minimum, we seek to obtain secure and dynamic configuration of available resources, then providing interactive access to these resources through web interfaces. These resources can be involved in tasks such as parallel computing, internetworking simulation laboratories, etc. Two test cases, representative of two broad use case classes, are implemented.

Keywords: Grid, Resource Management, High Performance Computing, Virtual Laboratory, Distance Education.

1 Introduction

Availability and integration of geographically spersed technological resources is currently one of the most important challenges. The quest for this kind of solutions calls for greater software and hardware requirements. A greater complexity in administration is also brought in, as resources across distributed environments are heterogeneous and a security posture is to be kept. Physical and virtual resources are logically grouped into Virtual Laboratories. These are defined by UNESCO[1] as electronic workspaces to collaborate and experiment in research and other creational activities, for generating and delivering research results using distributed information technologies. The biggest motivations for Remote Laboratories are their ability to scale up, to globally integrate organizations, to allow for sharing of specific resources, for collection and analysis of geographically distributed data, and for interdisciplinary specialists to cooperate. Among common uses for Virtual Laboratories are remote monitoring of production processes, remotely assessing performance of real and simulated facilities, remote configuration and management, and distance education applications. This work proposes a solution for the generation of Virtual Laboratories, using physical and virtual devices deployed at distinct local organizations and logically grouped by a virtual organization over a Grid environment. This must allow for dynamic and flexible configuration of a workspace by means of open standards and protocols.

We will describe the solution's architecture and components, and two use cases will be analyzed: a parallel computing environment (PARCOMP) and an educational remote virtual laboratory in internetworking (NETLAB).

2 Architecture Overview

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

A diagram of this architecture (fig. 1) pictures clients accessing a virtual organization through Internet. This particular Virtual Organization is composed by four Grid nodes, distributed into two physical organizations A and B. Both organizations own different types of Grid-available resources, but they both have dedicated clusters allowing the instantiation of virtual machines.

The solution we present here produces a space where independent virtual networks can be created. Administration of these networks runs completely apart from the physical networks that support them. Coordination tasks among local administration people are kept to a minimum.

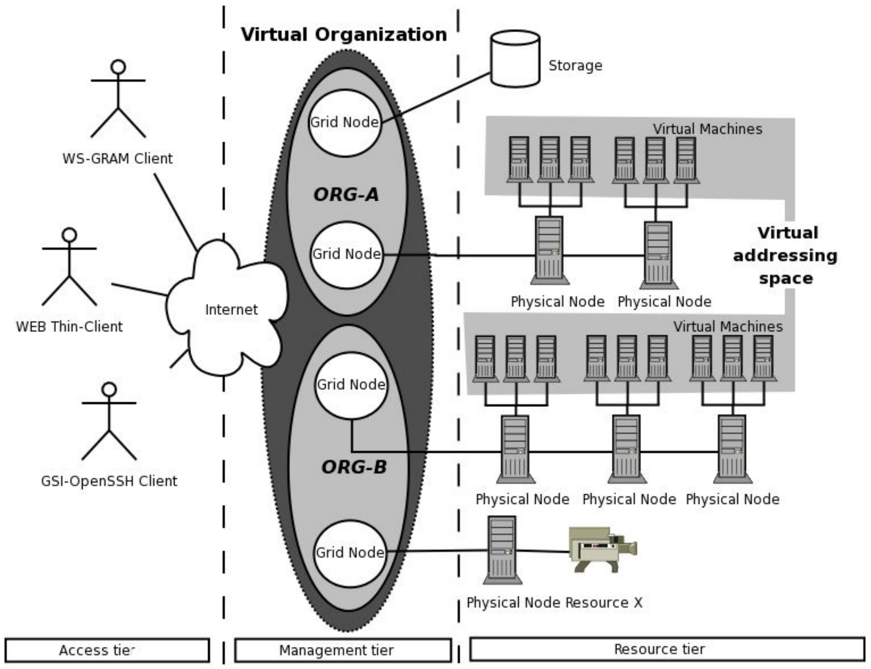


Fig. 1. Architecture

The first type of clients are Grid clients, which give access to resources at one or more Grid nodes from a command line interface. Grid nodes and their resources are managed with a Grid client. Web clients, having no Grid-specific tool or library to execute their tasks, are a second type. They do not know about implementation issues. Final users of the system sit at these clients, and their workspace is a virtual space. Finally, secure clients accesing remote terminals use a Grid security model. This is a special client of the first type. Teachers, or Grid administrators, use this kind of client.

The modules managing the Grid platform are found at the second tier. Each Grid node act as a gateway to the local cluster's private address space. Tasks execution in this space is always done through these nodes. To do work upon physical and virtual machines, command files can be executed in the domains managed by these Grid nodes.

At the third tier, the system's physical and virtual nodes are found. Physical nodes are dedicated to services requested by the Grid node. Each of these physical nodes can instantiate one or more resources such as virtual machines or simulated internetworking devices. Moreover, special resources like physical devices or persistent storage can be accessed from the virtual addressing space.

3 Architecture Components

We schematize in fig. 2 how a client accesses the system services involved, for instance, in the execution of a given request for computing nodes. Here the client calls a scheduling service, parametrically expressing his computing needs in an XML-based description language. The result from this stage specifies which physical nodes will be involved in the service, how many virtual nodes will be spawned at each of them, and which implementation of virtual machines will support each virtual node. These results are then sent by the scheduler as a parameter to the corresponding Web Service at each Grid node. Other parameters included in the scheduler's results are the range of network addresses to be used; and the service access points, if interactive access is in order. Each Grid node will instantiate the machines and return a service access locator for the client to access this network of virtual resources.

3.1 Requirements Specification

The system's input, at the edge next to the user, is a description of the virtual scenario the user intends to instantiate. The feasibility of this virtual scenario will be ensured, and then the proper logical resources will be mapped over a set of physical resources. Ideally, the user will operate an interface for edition of requirements to build the original description. For simplicity, to test our architecture we have devised a small programming interface which suffices to demonstrate the system's usage. This programming interface implements a simple set of classes modeling the desired devices and topology, as well as their mapping to the devices in the physical plant.

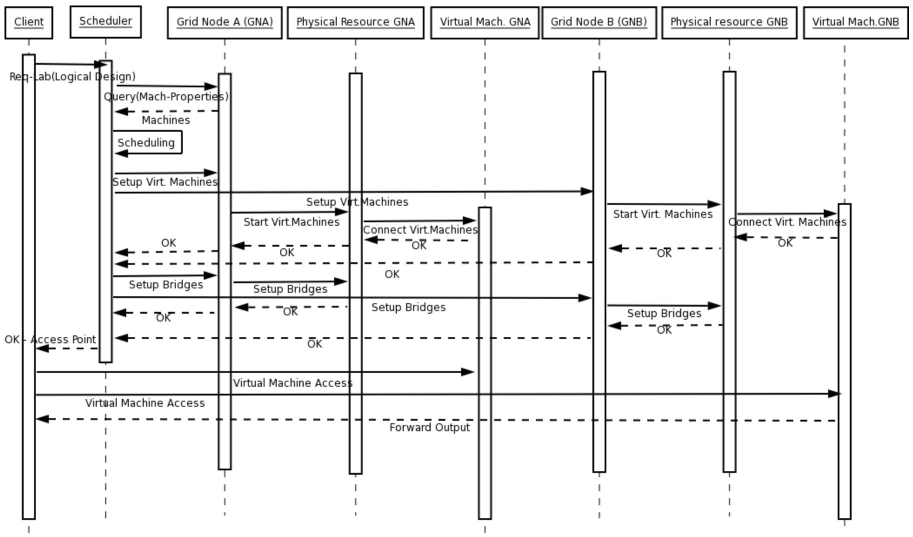


Fig. 2. Sequence Diagram

Using an object based language, the user can describe her scenario, essentially a graph whose nodes are the virtual nodes in the logical network to be instantiated; and whose arcs are the links that build up the desired topology. The scheduler will later resolve the request by mapping the virtual elements in the scenario to certain elements in the physical plant.

The programming interface assisted us during the tests to manually specify suitable scenarios and their mappings. This compensated for the lack of a visual interface and allowed to bypass the scheduling stage at the same time.

3.2 Scheduler

Grid nodes have status information about hosts in the physical plant. The scheduler, by reasoning upon configuration and status data about those physical hosts, will output an allocation plan or mapping between virtual and pyhsical nodes. A given Grid node will offer a variable but limited amount of resources. Under non-availability or high load conditions, the scheduler can instantiate logical resources at several other Grid nodes if needed, then building a virtual network among them. The ability to detect which physical machines in the Virtual Organization have virtualization software is needed to carry on the scheduling phase.

In addition to building the mapping plan, the scheduler must return the service access locators for the virtual elements instantiated and other physical devices or services. For the PARCOMP use case, the (*sparse*) mapping will instantiate no more than a single virtual machine per physical host; all these virtual machines will share the same addressing space; and the output to the user will be just a single service access locator and a service URI for persistent storage. On the other hand, for the NETLAB use case, the (*dense*) mapping will work under fundamentally different scheduling constraints, i.e. possibly requesting many virtual resources per physical host. These virtual resources will also live in the same addressing space but the scheduler's output to the user will be a set of service access locators and URIs, one for each virtual resource instantiated.

For the NETLAB use case, physical topology can be exploited according to logic topology (i.e. mapping certain virtual links to certain physical links). In the PARCOMP use case, it makes little sense to instantiate more than one virtual machine on the same physical machine, as the application is performance-driven; and the whole instantiated scenario should be located into a same physical cluster if possible.

3.3 Logical to Physical Mapping

The monitoring and discovery system offered by the Grid environment allows users to know which resources are considered a part of the Virtual Organization, and to monitor their status. Every piece of information acquired through aggregation services is maintained in XML and accessed through XPath queries. The same can be said about other query mechanisms through Web Services.

In our work, the information flow hierarchy begins at the cluster monitoring level [2]. The Ganglia package was used to link cluster data to each one of the Grid nodes. To acquire information about active machines in the clusters, the scheduling service sends an XPath query to the Grid node. This query is generated following the requirements presented in the logical scenario and can contain threshold specifications such as free memory or idle CPU status.

3.4 Resource Instantiation

Once a feasible logical-to-physical mapping is obtained, and having previously ensured the availability of resources, virtual machines are created in physical nodes under each Grid node. As shown in fig. 2, the scheduler will send allocation requests to every Grid node using a Web Services interface. The Grid nodes in turn are to redirect these requests to corresponding physical nodes. Scripts were implemented to translate the specifications for virtual machine instantiation into configuration files to be interpreted in each physical node. Extending these scripts transparently allows for the use of new virtualization technologies.

3.5 Network Virtualization

After every logical resource requested in the original scenario is up and running, they have to be interconnected so as to reflect the required topology. However, we seek to hide the details concerning the particular networks and physical hosts existing behind each Grid node. Therefore, we need a private network to relate the virtual machines. We selected VDE_SWITCH [3] and NetCat[4] as tools to deploy virtual "cabling". Custom scripts were created to build up a solution according to the system's needs. Virtual machines are linked by means of "virtual cables" using TCP tunnels as a transport between physical nodes. A similar approach is used to interconnect LANs on different domains, this time under an SSH connection to ensure privacy. Level 2 frames exchanged by the virtual machines are encapsulated into the tunnel. The elements in the instantiated scenario stay transparently interconnected over a private virtual addressing space which cannot collide with other addressing schemes at the participating Grid nodes.

4 Use Case Implementation

The test cases selected for this work have different goals, and their corresponding scenarios present different topologies and interaction requirements. In the NETLAB test case, links do have specific attributes, such as bandwidth, delay or reliability, and they are crucial to the functioning of the instantiated scenario. Access to the instantiated elements by the user is essentially interactive. In the PARCOMP case, the effective processing power of virtual machines is what matters. Tasks are launched in batch mode, and they are not interactive. As a consequence, both scenarios will challenge the mapping process with different goals and constraints.

Our computing environment is composed by two clusters. One of them is located at premises of the Universidad Nacional del Comahue (UNC) and the other one is hosted at a local IT company (CDF). Both locations are in the same city. Hosts in the UNC cluster are Pentium IV 2.2 MHz machines with 512 MB RAM. The CDF cluster is composed by five Pentium IV 3.06 GHz, HyperThreading machines, with 1GB RAM. The software used for managing the Grid environment is Globus Toolkit 4 [5]. The operating system was Linux, distributions CentOS and Fedora Core 6. The PARCOMP use case was implemented on Xen 3.0.3 [6] virtual machines. For the NETLAB use case, QEMU [7] virtual machines were also used.

4.1 Parallel Application Use Case (parcomp)

The goal for the PARCOMP use case is testing the practical feasibility, and measuring the overhead, of our solution for the execution of parallel applications. We want to benefit from a Grid environment's capabilities, while not having to modify the application. For this use case we selected a parallel application used to model transmission in neural synapses. This application belongs to former work developed by the Complex Systems research group at Universidad Nacional General Sarmiento[8]. The application works under the master-worker paradigm, adequately balancing load among the workers. After each Worker finishes a batch of work, it reports partial results to the Master. The Master then collects the partial results and proceeds with other sequential processing [9].

The application was run using the MPICH 1.2.7p1 parallel library. Three kinds of tests were performed. The first test consisted in the execution of the application without involving any virtualization. The goal of this first test was to acquire a baseline against which virtualization overhead could be measured. The next test uses virtual machines over physical machines, but no network virtualization. This test allows us to observe application processing overhead. The last execution test uses virtual machines as well as virtual networking, and allows us to know the amount of communications overhead in the complete solution.

As can be seen in Table 1, the Worker's computing overhead for the virtual computing-only test is very low. However, measurements taken at the Master (which account for coordination and data exchange with Workers) show a noticeable increment in computing time for the different virtualization types.

Table 1. Parallel Aplication Benchmark

	Master	Worker
Pure Parallel	512 secs.	50 secs.
Virtual Machines -Physical Network	640 secs.	69 secs.
Virtual Machines - Virtual Network (With Netcat)	768 secs.	73.5 secs.
Virtual Machines -Virtual Network (With SSH)	768 secs.	74 secs.

4.2 Internetworking Laboratory Use Case (netlab)

The NETLAB use case consists in a computer internetworking scenario. Students are expected to practice some network configuration and administration techniques inside this environment. We assume resources are limited at the university labs, hence machines hosted at remote facilities owned by an external entity (such as a partner company) are made available under an agreement and accessed through Internet. To this end, the partner entity stores a repository of system images that can be used to do networking practices. The goal for this use case is to show how the proposed infrastructure can allow the automatic instantiation of virtual laboratory scenarios using a Grid environment and making use of physical resources available at each organization.

As a simple example, the practice assignment implies interconnecting three nodes, located on two different LANs linked by a router. First, the teacher generates a template with the logical design of the assignment. The particular amount of physical resources available at the university labs is immaterial at this stage. Then she invokes the scheduling service of the solution for every student group taking the assignment. The scheduler will instantiate virtual machines on physical hosts across the virtual organization as needed. The virtual scenarios in execution will be the resulting workspaces. The virtual devices are made available through their service access locators, published on web pages dynamically generated. The students, using a Java-enabled browser, access the resources from different locations and collaborate on the assignment.

The experiences showed variable delays, depending on the link capacity and offered load at the moment of performing the assignment work. Performance measurements were done using the NetPerf tool[10]. As shown in fig. 3, the non-virtualized execution, at 98 Mbps, is close to the theoretical expectation. As connections are virtualized, transmission capacity loss reaches about 50% over an SSH “cable”.

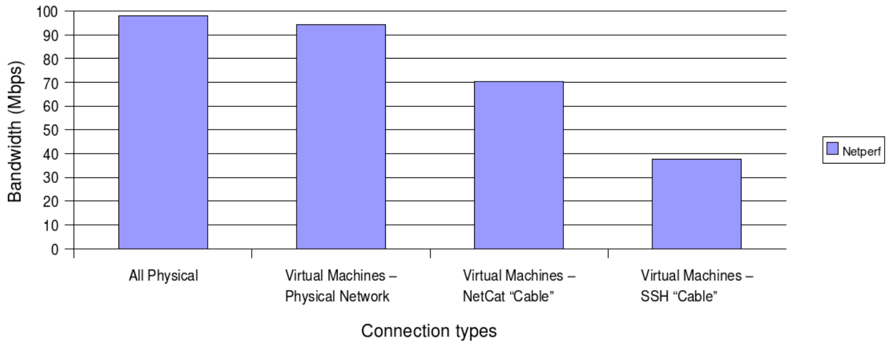


Fig. 3. Virtual Network Benchmark

5 Related Work

Some amount of work has been published about Virtual Laboratories, usually showing high performance environments as a use case, and highlighting some aspect of the solution. The main concern of the Virtual Spaces project [11] is defining and administrating virtual spaces in a Grid environment. Its use case is based on clusters of virtual machines inside a local area network. Cluster on Demand[12] implements the packaging of a cluster scheduler to obtain subsets of a physical cluster through dynamic network address assignment. VioCluster[13] is closely related to this work, although it does not consider Grid for virtual network configuration, nor dynamic discovery of candidate physical machines for virtual machine instantiation. Instead, VioCluster focuses on automatic negotiation of administration domains following established policies, and relates to autonomic concepts. In-VIGO[14], at a very higher abstraction level, allows applications to use virtual environments through Grid services.

Our work can be compared to these efforts, as all of them seek to provide virtual environments through the use of virtual machines. Our approach differs from them in that Grid technology is used to build the communication infrastructure for resources, allowing the dynamic creation of common spaces across different domains.

6 Conclusion

We described a feasible solution for the coordinated usage of geographically spersed computing resources, under a virtual organization scheme provided by the Grid infrastructure. The solution makes use of virtual machines for flexibility and transparency.

Our work delivers an approach to issues related to configuration, access and resource management. Regarding configuration, a simple object-based language has been specified for the design and validation of the logical requirements. As for access, a model for service access locators, achieving virtual terminal redirection from the resources across Internet, onto the user, has been developed and tested. As for management, Grid middleware functionality has been used to allow remote execution and data transfer across administrative domains with no work required from local administrators.

Use case implementation allowed us to gather some experience about the proposed solution's behaviour under different sets of requirements. The first one, the use of a parallel application, where network configuration is trivial, with a single type of virtual machine but with strong performance requirements. The second one, a remote laboratory where the main requirement is flexibility in the configuration of several kinds of scenarios, and there is a demand for a high number of virtual machines, but no strong performance requirements.

The implementation of these two use cases motivated a detailed analysis. The existence and separate configuration of the private addressing space provided addressing transparency and allowed the enforcement of a security policy when working on lab assignments (NETLAB). Usage of virtual machines, when implementing a parallel computing application (PARCOMP), eased working with divergingly configured clusters (such as when having different versions of parallel libraries) with little administration overhead. In both cases, introducing a Grid environment into the problem of cluster hosts configuration allows for the combination of a greater number of resources on demand, therefore enhancing usage patterns.

In the case where performance requirements are not specific (NETLAB), and where primary importance is given to the creation of a virtual work environment, separated from the encompassing physical environment, this is an acceptable solution which allows users to make an efficient usage of available resources, or introduce otherwise unavailable elements. When the application bears performance constraints (PARCOMP), usage analysis must lead to further studying the infrastructure behaviour for the particular application (namely, computing power demand, amount of coordination messaging, bulk data transfer volume, network virtualization overhead must all be taken into account).

Our future work includes studying the optimization of the virtual machines' networking. Enhancing performance at the point of access to the network is of prime importance for the parallel computing class of use cases. For Xen virtual machines, this is currently dependent on Linux "bridge" devices. As for configuration, we plan to extend the specification language, introducing new interfaces. As for administration, we plan to define a framework for querying virtual scenarios and for dynamic creation of virtual machines on a package- or service-provided basis, so as to be able to easily manage repositories to enhance the solution's availability and flexibility.

References

1. James, P., Vary, E.: Report of the Expert Meeting on Virtual Laboratories. Technical Report CII-00/WS/01, International Institute of Theoretical and Applied Physics (IITAP), UNESCO (2000)
2. Sacerdoti, F.D., Katz, M.J., Massie, M.L., Culler, D.E.: Wide area cluster monitoring with ganglia, pp. 289–298 (2003)
3. Davoli, R.: Vde: Virtual distributed ethernet. In: TRIDENTCOM 2005: Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM 2005), Washington, DC, USA, pp. 213–220. IEEE Computer Society, Los Alamitos (2005)
4. NetCat, `http://netcat.sourceforge.net/`
5. Foster, I.T.: Globus toolkit version 4: Software for service-oriented systems. In: NPC, pp. 2–13 (2005)
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 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164–177. ACM Press, New York (2003)
7. QEMU, a fast and portable dynamic translator (2005)
8. Carusela, M., Perazzo, R., Romanelli, L.: Information transmission and storage sustained by noise. Physica D 168–169, 177–183 (2002)
9. Argollo, E., Gaudiani, A., Rexachs, D., Luque, E.: Tuning application in a multi-cluster environment. In: Euro-Par, pp. 78–88 (2006)
10. NetPerf., `http://www.netperf.org/netperf/`
11. Foster, I.T., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B., Zhang, X.: Virtual clusters for grid communities. In: CCGRID, pp. 513–520. IEEE Computer Society, Los Alamitos (2006)
12. Chase, J.S., Irwin, D.E., Grit, L.E., Moore, J.D., Sprenkle, S.E.: Dynamic virtual clusters in a grid site manager. In: HPDC 2003: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC 2003), p. 90. IEEE Computer Society, Los Alamitos (2003)
13. Ruth, P., Mcgachey, P., Xu, D.: Viocluster: Virtualization for dynamic computational domains. In: Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2005) (2005)
14. Adabala, S., Chadha, V., Chawla, P., Figueiredo, R., Fortes, J., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L., Zhu, X.: From virtualized resources to virtual computing grids: The in-vigo system. Future Gener. Comput. Syst. 21(6), 896–909 (2005)

Information Service of Virtual Machine Pool for Grid Computing

Marcel Kunze and Lizhe Wang

Institute for Scientific Computing, Forschungszentrum Karlsruhe
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
{Marcel.Kunze,Lizhe.Wang}@iwr.fzk.de

Abstract. Distributed virtual machines can help to build scalable, manageable and efficient Grid infrastructures. The work proposed in this paper focuses on employing virtual machines for Grid computing. In order to efficiently run Grid applications, resource information of virtual machines should be provided. The paper firstly discusses the system architecture of virtual machine pools and the process of information retrieval from virtual machines. Based on the characterization of the system model, the paper presents the work how to retrieve resource information from Xen/VMware virtual machines via VMware CIM SDK and light weight Java agents. The resource information is integrated into Grid information service. The work is implemented in a test bed with Xen/VMware virtual machines and Globus Toolkit.

Keywords: Virtual machine, Grid computing, Information service.

1 Introduction

Grid computing technology [1] offers interesting solutions for parallel and distributed computing. It can provide reliable, collaborative and secure access to remote computational resources as well as distributed data and scientific instruments. However, as more resources are shared in Grid environments largely sized distributed data can be manipulated and applications for Grid computing may become complex and intractable. Grid workflows nowadays are frequently used to model complex applications for Grid computing. Based on Grid middleware and Grid infrastructures a Grid workflow system defines, specifies and manages a workflow on computational Grids [2].

A virtual machine is a computing platform that creates a virtualized layer between the computing hardware and the application. This paper is devoted to discuss a Grid workflow system on distributed virtual machines. There are advantages like on demand creation and customization, performance isolation, legacy software support and ease of management.

We try to build a workflow system based on virtual machines [3]. A Grid workflow user could experience the following challenges:

- Site autonomy
  In the virtual machine based Grid system the hosting resources are usually owned and controlled by different organizations at different sites. Users may have to meet different resource management policies during the manipulation of their virtual machines.
- Hierarchy
  The virtual machine based Grid system is hierarchical in nature. It contains several levels, for instance, virtual machine level, hosting resource level and the user access point, i.e., Grid portal.
- Heterogeneity
  The virtual machine based Grid system includes heterogeneous hosting resources, virtual machine technologies (e.g., Xen, VMware) as well as programming interfaces.
- Large scale distribution
  Computer centers and data centers frequently employ virtual machines and build Grid infrastructures across geographically distributed sites.

The paper implements information collectors in Xen/Vmware virtual machines and builds Grid information service with virtual machine information providers. The paper is organized as follows: related work is investigated in Section 2; Section 3 presents the system architecture of employing virtual machines for Grid computing and Section 4 provides details of the design and implementation of the system. In Section 5, test results are discussed; Section 6 concludes the paper and points out future work.

2 Related Work

Since several years the Grid computing research community shows interest for virtual machines and virtual environments. The typical Virtual Machine Monitor (VMM) or hypervisor setup includes Xen VMM [4], VMware workstation and ESX server [5], and User Mode Linux [6].

The Globus alliance recently implemented the concept of virtual workspace [7], which allows a Grid client to define an environment in terms of its requirements, manage it, and then deploy the environment on the Grid. The implementation is based on Globus Toolkit 4 (GT4) and it only supports Xen VMM. Some other research work also focuses on deploying computing systems or test beds with virtual machines, for example, virtualization of batch queueing system [8], Grid-Builder [9], using virtual machine as Grid gateway [10], multi-site MPI platform with Xen virtual machine [11], migration of virtual machines in MAN/WAN [12].

Other researchers try to build virtulized middleware for clusters and distributed systems. Xen Grid Engine [13] employs an approach to create dynamic virtual cluster partitions using para-virtualization techniques. The work presented in [14] builds virtual clusters and virtual networks for applications in large distributed systems. The In-VIGO [15] project aims to build virtulization middleware for computational Grids. In-VIGO provides a distributed environment where multiple application instances can coexist in virtual or physical resources such that clients are unaware of the complexity inherent to Grid computing.

Another important topic is the performance analysis of virtual machines or virtual environments. The Xen group brings a performance evaluation and comparison between serval popular VMMs concerning the performance overhead in different scenarios [16]. Other research efforts refer to virtual machine based systems, i.e., performance of para- and paene- virtualized systems [17], performance enhancement of SMP clusters with virtual machines [18].

3 System Architecture of Employing Virtual Machines

This section describes the system architecture of a virtual machine based system for Grid computing. The system architecture is defined hierarchically:

– Grid level
  The target Grid system contains multiple geographically distributed sites, which could be computer centers, data centers and research institutes. On the Grid level, each site is represented and accessed via a single access point, where Grid middleware, for instance Globus Toolkit, is installed.
– Site level
  Each site provides a number of hosting resources, for instance cluster, PVP or MPP. Physical resources at each site are interconnected and can support multiple virtual machines.
– Virtual machine level
  Hosting resources are installed with VMMs, such as VMware ESX server or Xen hypervisor, and back several virtual machines. Multiple virtual machines can be grouped into resource pools.

In the system model defined in Fig. 1, each site in the Grid system provides an access point (or head node), via which clients can manipulate the virtual machines backed by the Virtual Machine Monitor (VMM), like for instance Xen VMM, VMware server or VMware ESX server. For example, users can deploy virtual machines in Grid infrastructures, retrieve resource information of virtual machines and submit Grid applications in workflow to virtual machine pools. The access point can be installed using Globus Toolkit or Condor daemon, virtual machines can thus be configured as a cluster with a GRAM (Globus Resource Allocation Manager) head node. The Grid application is submitted to virtual machine pools via GRAM.

4 Design and Implementation of an Information Service

4.1 Overview

The information service consists of an information collector in virtual machines, an information provider in the access point, and the aggregated Globus index service. The information collector which runs inside a virtual machine is used to

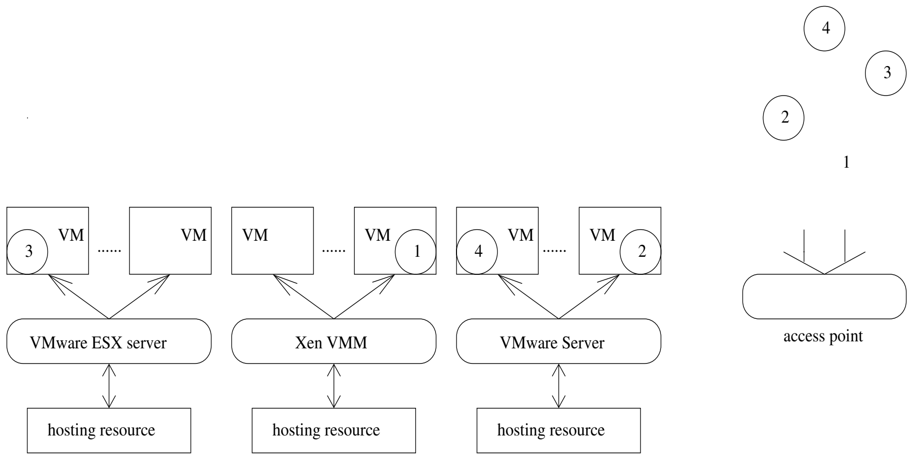


Fig. 1. System model for virtual machine based Grid computing

retrieve resource information for the information provider. Information providers for Globus MDS 4 (Monitoring and Discovery System) organize the resource information from information collectors in predefined XML schema, which are aggregated into the Globus index service. The WebMDS can be configured as a graphical user interface based on the Globus index service (see also Fig. 2).

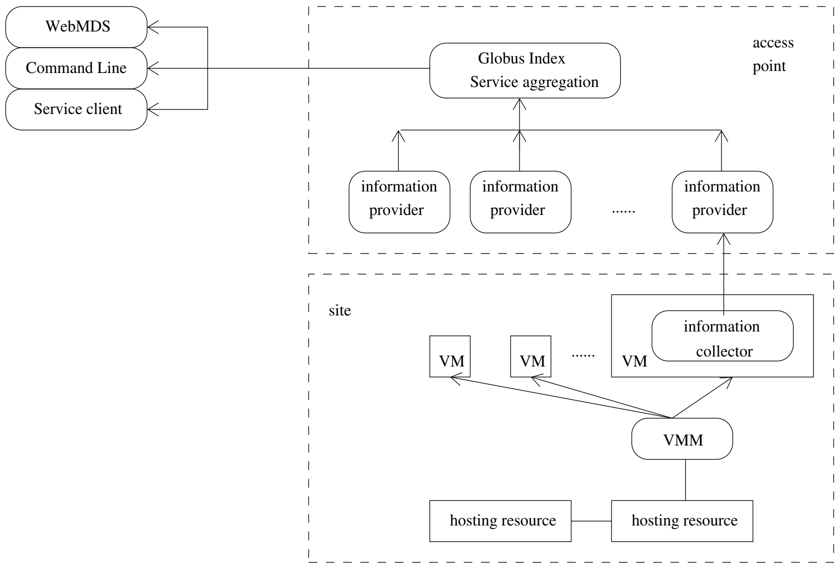


Fig. 2. Overview of the information service

4.2 Information Collector

The information collector is a light weight software, which resides inside a virtual machine and collects resource information. In the system, two types of information collectors have been implemented:

– CIMOM for VMware ESX server
  VMware ESX server is a commercial virtualization product of VMware Inc. Common Information Model (CIM) [19] is defined as an international standard by the Distributed Management Task Force (DMTF) [20]. The VMware ESX server together with CIM SDK provides a CIM-compliant object model for virtual machines and their related storage devices. Fig. 3 shows a typical configuration environment of VMware ESX server. The virtual machine contains a virtual disk that resides as a virtual disk file on a storage area network.

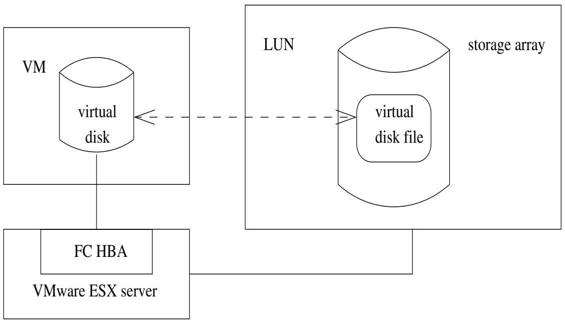


Fig. 3. Sample environment of VMware ESX server

  The SMI-S (Storage Management Initiative Specification) schema for the sample ESX server environment is shown using UML in Fig. 4. *ESX-ComputerSystem* is the kernel object of the system. It associates *VM*, *VirtualDisk* and *FC HBA&LUN*[1] with *HostedDependency*, *HostedStoragePool* and *SystemDevice* relationships respectively. *VM* is associated with *VirtualDisk* with *ArchiConnection* relationship. The latter is associated with *FC HBA&LUN* in *ConcreteComponent* relationship.

  The pegasus CIMOM (CIM Object Manager) [21] is installed on the VMware ESX server. The information provider at the access point works as the CIM client, communicates with pegasus CIMOM to retrieve information from virtual machines and their associated storage. The information provider complies with SMI-S profile [22] and transports CIM XML over HTTP/HTTPS to pegasus CIMOM (see also Fig. 5).
– Light weight Java agent for VMware server and Xen VMM

[1] Fibre Channel Host Bus Adaptor & Logical Unit Number

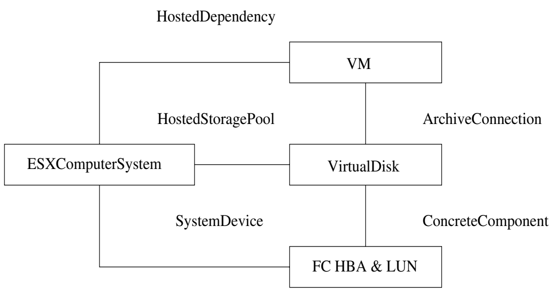


Fig. 4. CIM schema for VMware ESX server

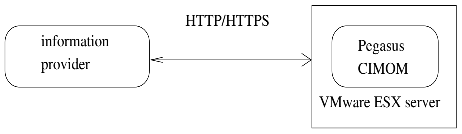


Fig. 5. CIMON for VMware ESX server

VMware server is a free virtualization product of VMware VMM without CIM SDK support. The Xen CIM project is providing an implementation of the preliminary virtualization and resource allocation models, which is currently being defined by the DMTF System Virtualization, Partitioning, and Clustering Working Group (SVPC WG). As the CIM SDK or programming support is not available for Xen VMM and VMware server we implemented a light weight Java agent as information collector for Xen VMM and VMware server.

The information agent contains the following components (see also Fig. 6):

- Information Sensor

  The information sensor is designed to retrieve certain types of resource information. For example, a memory sensor provides memory information. In the implementation, information sensors are perl scripts to retrieve memory, CPU and OS information.
- Information Engine

  The information engine forwards the information collected from information sensors to the access point. The information engine firstly queries on the information item cache in the memory. When the information in the cache is missed or expired, the engine then invokes information sensors to deliver the information.
- Information Item Cache

  The information item cache is a block of memory allocated to store values of information items. Each time after the information sensors are executed, the information item cache is updated.

The communication between the information agent and information provider in the access point is implemented using TCP/IP socket communication. The information agent periodically updates the information to the information provider at the access point. The information provider can also query information agents to obtain the latest information.

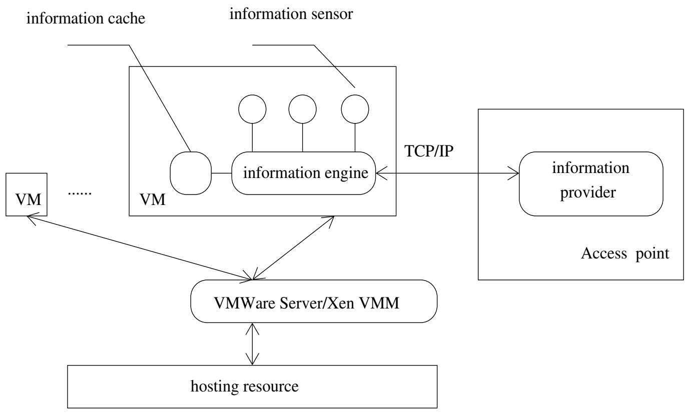


Fig. 6. Light weight Java agent for VMware server and Xen VMM

4.3 Information Provider for Globus Index Service Aggregation

An information provider resides on the access point and provides virtual machine information to GT4 index service back end [23]. The information retrieved from virtual machines are organized in plain text format. The information provider defines a simple XML schema and reorganizes the plain text based resource information in the schema.

It is also demanded to configure Globus Toolkit to enable the information provider for Globus index service, for example, registering the information provider, mapping the information provider in the deployment file of the Globus index service. Globus index service clients can get resource information from virtual machines, for example, via WebMDS [24] or from the Globus Toolkit command line.

5 Test Results

5.1 Test Bed

The actual test bed is configured as shown in Fig. 7. **Blade10**, **Blade11** and **Lizhe3** are hosting resources which are installed with Xen VMM, VMware ESX server and VMware server, respectively. **VM1**, **VM2** and **VM3** are virtual machines backed by the hosting resources that form the virtual machine pool. **Lizhe2** is the access point for the virtual machine pool (see also Tab. 1).

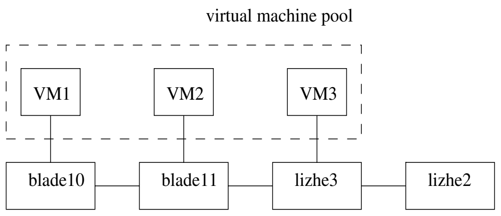


Fig. 7. Test bed

Table 1. Test bed summary

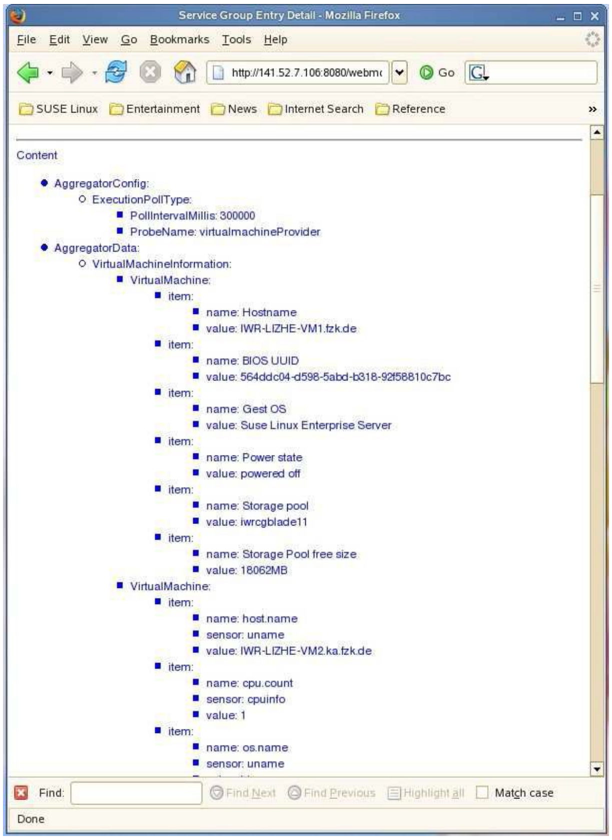
Resource Name	Resource Type	Software installed
Blade10	IBM BladeCenter HS20	Xen VMM
Blade11	IBM BladeCenter LS20	VMware ESX server
Lizhe3	Linux Workstation	VMware server
Lizhe2	Linux Workstation	GT4, Condor, GridFTP
VM1, VM2, VM3	Virtual Machine	—

5.2 Test Results

The access point runs an information provider to retrieve information from **VM1/VM3** and **VM2** via light weight Java agent and VMware ESX server SDK respectively. The information provider collects the resource information and organizes it with XML schema (shown as below):

```
<VirtualMachineInformation>
<VirtualMachine>
<item name="Hostname" value="IWR-LIZHE-VM2.fzk.de"/>
<item name="BIOS UUID"
        value="564ddc04-d598-5abd-b318-92f58810c7bc"/>
<item name="Gest OS" value="Suse Linux Enterprise Server"/>
<item name="Power state" value="powered off"/>
<item name="Storage pool" value="iwrcgblade11"/>
...
</VirtualMachine>
<VirtualMachine>
<item name="host.name" value="IWR-LIZHE-VM1.ka.fzk.de"
        sensor="uname"/>
<item name="cpu.speed" value="1000.353" sensor="cpuinfo"/>
<item name="os.name" value="Linux" sensor="uname"/>
<item name="mem.total" value="254552" sensor="meminfo"/>
...
</VirtualMachine>
...
</VirtualMachineInformation>
```

The information provider thus furnishes the organized information to the back end of Globus index service. Users can retrieve the information with client programs of Globus index service or browse the information via WebMDS. Fig. 8 shows the resource information retrieved from virtual machine pool with WebMDS on **Lizhe2**. These results justify the prototype implemented on the test bed.


Service Group Entry Detail - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://141.52.7.106:8080/webm
Go
SUSE Linux Entertainment News Internet Search Reference
Find: Find Next Find Previous Highlight all Match case
Done

Content

- AggregatorConfig:
  - ExecutionPollType:
    - PollIntervalMillis: 300000
    - ProbeName: virtualmachineProvider
- AggregatorData:
  - VirtualMachineInformation:
    - VirtualMachine:
      - item:
        - name: Hostname
        - value: IWR-LIZHE-VM1.fzk.de
      - item:
        - name: BIOS UUID
        - value: 564ddc04-d598-5abd-b318-92f58810c7bc
      - item:
        - name: Gest OS
        - value: Suse Linux Enterprise Server
      - item:
        - name: Power state
        - value: powered off
      - item:
        - name: Storage pool
        - value: iwrcgblade11
      - item:
        - name: Storage Pool free size
        - value: 18062MB
    - VirtualMachine:
      - item:
        - name: host.name
        - sensor: uname
        - value: IWR-LIZHE-VM2.ka.fzk.de
      - item:
        - name: cpu.count
        - sensor: cpuinfo
        - value: 1
      - item:
        - name: os.name
        - sensor: uname

Fig. 8. Resource information from virtual machines

6 Conclusion and Future Work

Virtual machines are widely accepted in computer centers to support various applications. This paper implements an information service of virtual machine pools in a computer center for Grid computing. The information service can monitor virtual machines backed by popular VMMs, such as Xen VMM, VMware server and VMware ESX server. The paper furthermore builds interfaces of virtual machine pools for computational Grids. The next step is to test the performance with some real applications of high energy experiments, like for instance a CMS benchmark [25].

References

1. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Philosophy of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Open Grid Service Infrastructure Workgroup, Global Grid Forum (2002)
2. Wang, L., Jie, W., Zhu, H.: State-of-Arts: Workflow Management for Grid Computing. In: Grid Technologies: Emerging from Distributed Architectures to Virtual Organizations, ch. 9, pp. 241–271. WIT Press (2006)
3. Wang, L., Kunze, M.: On the Design of Virtual Environment Based Workflow System for Grid Computing. In: Proceedings of International Workshop on Workflow System for Grid Computing and Applications (WSGE), China, October 2006, pp. 212–218. IEEE Computer Society Press, Los Alamitos (2006)
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization . In: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), pp. 164–177. ACM Press, New York (2003)
5. VMware virtualization products, `http://www.vmware.com`
6. User Mode Linux, `http://user-mode-linux.sourceforge.net`
7. 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)
8. Buege, V., Kemp, Y., Kunze, M., Oberst, O., Quast, G.: Virtualizing a Batch Queueing System at a University Grid Center. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 397–406. Springer, Heidelberg (2006)
9. Childs, S., Coghlan, B., McCandless., J.: GridBuilder: A Tool for Creating Virtual Grid Testbeds. In: Proceedings of 2nd IEEE Conference on eScience and Grid computing (e-Science), pp. 77–77. IEEE Computer Society Press, Amsterdam, Netherlands (2006)
10. Childs, S., Coghlan, B., O'Callaghan, D., Quigley, G., Walsh., J.: A Single-computer Grid Gateway Using Virtual Machines. In: Proceedings of the 19th International Conference on Advanced Information Networking and Applications, pp. 310–315. IEEE Computer Society Press, Washington, DC, USA (2005)
11. Tatezono, M., Maruyama, N., Matsuoka, S.: Making Wide-Area, Multi-Site MPI Feasible Using Xen VM. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 387–396. Springer, Heidelberg (2006)
12. Travostino, F., Daspit, P., Gommans, L., Jog, C., de Laat, C., Mambretti, J., Monga, I., van Oudenaarde, B., Raghunath, S., Wang, P.: Seamless Live Migration of Virtual Machines over the MAN/WAN. Future Generations Computer Systems 22, 901–907 (2006)
13. Fallenbeck, N., Picht, H.J., Smith, M., Freisleben, B.: Xen and the Art of Cluster Scheduling. In: Proc. of 1st International Workshop on Virtualization Technology in Distributed Computing, IEEE Computer Society Press, Los Alamitos (2006)
14. Ruth, P., Jiang, X., Xu, D., Goasguen., S.: Towards Virtual Distributed Environments in a Shared Infrustructure. IEEE Computer 38(5), 63–69 (2005)

15. Adabala, S., Chadha, V., Chawla, P., Figueiredo, R., Fortes, J., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L., Zhu., X.: From Virtualized Resources to Virtual Computing Grids: the In-VIGO System. Future Generation Computing Systems 21(6), 896–909 (2005)
16. Performance Comparison of VMMs (July 2006), http://www.cl.cam.ac.uk/research/srg/netos/xen/performance.html
17. Soltesz, S., Fiuczynski, M.E., Peterson, L., McCabe, M., Matthews, J.: Virtual Doppelganger: On the Performance, Isolation, and Scalability of Para- and Paene-Virtualized Systems (November 2005), http://www.cs.princeton.edu/~mef/research/paenevirtualization.pdf
18. Strazdins, P., Alexander, R., Barr, D.: Performance Enhancement of SMP Clusters with Multiple Network Interfaces Using Virtualization. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 452–463. Springer, Heidelberg (2006)
19. Common Information Model, http://www.dmtf.org/standards/cim
20. Distributed Management Task Force, http://www.dmtf.org
21. Pegasus CIMOM, http://www.openpegasus.org
22. Storage Management Initiative Specification, http://www.snia.org/smi/tech_activities/smi_spec_pr/spec
23. Information Provider for Globus Index Service, http://www.globus.org/toolkit/docs/4.0/info/index/WS_MDS_Index_HOWTO_Execution_Aggregator.html
24. WebMDS, http://www.globus.org/toolkit/docs/4.0/info/webmds
25. Compact Muon Solenoid, http://cms.cern.ch

Virtual Cluster Management with Xen

Nikhil Bhatia and Jeffrey S. Vetter

Future Technologies Group
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA 37831
{bhatia,vetter}@ornl.gov

Abstract. Recently, virtualization of hardware resources to run multiple instances of independent virtual machines over physical hosts has gained popularity due to an industry-wide focus on the need to reduce the cost of operation of an enterprise computing infrastructure. Xen is an open source hypervisor that provides a virtual machine abstraction layer which is very similar to the underlying physical machine. Using multiple physical hosts, each hosting multiple virtual machines over a VMM like Xen, system administrators can setup a high-availability virtual cluster to meet the ever-increasing demands of their data centers. In such an environment, the Xen hypervisor enables live migration of individual virtual machine instances from one physical node to another without significantly affecting the performance of the applications running on a target virtual machine. This paper describes a scalable Virtual Cluster Manager that provides such application agnostic cluster management capabilities to the system administrators maintaining virtual clusters over Xen powered virtual nodes.

1 Introduction

Recently, virtualization of hardware resources to run multiple instances of independent virtual machines over physical hosts has gained popularity as a potential solution to an industry-wide focus on the need to reduce the cost of operations of enterprise and scientific computing infrastructures [1]. Industrial and academic installations of these clusters can contain thousands of physical nodes. Such clusters are prone to changes in the availability of physical resources due to unfortunate conditions like node failure due to overheating, or system administrative tasks like dynamic load balancing and preventative maintenance. Meanwhile, applications and users on these infrastructures expect highly available, reliable, and transparent operation.

Accordingly, there has been a prolific rise in the research and development of Virtual Machine Monitors (VMMs) that employ virtualization at different levels in the system software stack (e.g., Xen [2], QEMU [3], VMWare [4], User Mode Linux [5]). One such system exemplifying this trend is Xen: an open source hypervisor that provides a virtual machine abstraction layer which is very similar to the underlying physical machine. Xen's type of virtualization, often termed as para-virtualization, overcomes the typical performance loss due to virtualization by maintaining hardware information per individual VM inside the VMM interface. On the other hand, this para-virtualization requires substantial modification to the hardware dependent code in the target operating systems running over the virtual machine. This virtualization provides many benefits including the ability to save the entire VM to persistent storage, and then restart it later on the same, or on a *different,* physical host [6]. When using large pools of physical hosts, with each physical host containing multiple virtual machines, system administrators can easily construct a high-availability virtual cluster to meet the ever-increasing demands of their data centers and scientific computing clusters. With these virtualization capabilities, system administrators can handle various management tasks, such as dynamic load balancing of virtual nodes, and eviction of applications from a physical nodes to prepare for maintenance or to preempt a expected failure, transparently to the individual applications running in the virtual machines.

This paper describes a Virtual Cluster Manager that provides such application-agnostic cluster management capabilities to the system administrators maintaining these virtual clusters. We demonstrate its capabilities that include remote migration and dynamic load balancing of VMs across a pool of physical nodes. This framework manages a virtual cluster powered by the Xen virtual machines across multiple physical nodes. Our prototype-- the Xen Virtual Cluster Manager, henceforth referred to as XCM -- provides several of these features. First, it provides an overview of the performance of virtual machines to the system administrators on a per physical node basis. Second, it provides capabilities to initiate administrative tasks like automatic load balancing of VMs across physical resources based on their utilization and eviction of VMs from physical nodes in preparation for maintenance at arbitrary intervals.

Following is the organization of this paper. Section 2 describes the Virtual cluster organization. Section 3 describes the related work in this area. Section 4 describes the XCM framework which is divided into two parts: the XCM client and the XCM Daemon. Section 5 describes the implementation decisions we made during the course of this project. Section 6 provides the reader with an overview of our framework in action. Finally, we discuss out future research goals in this area in section 7 and conclude in section 8.

2 Virtual Clusters

Virtual clusters [7] [8] are comprised of several physical nodes running a virtual machine monitor (e.g. Xen) and hosting multiple virtual machines (often referred to as DOM Us) which in turn are running several user-level applications. Such an infrastructure can easily run into managing hundreds (or even thousands) of virtual machine instances running over tens (or hundreds) of physical nodes. These configurations are beneficial for large scale computing facilities because it reduces the cost of operation of these data centers by replacing the need for hundreds of physical servers by having hundreds of virtual machine over tens of physical servers.

Also, this infrastructure can also be useful in managing distributed memory cluster environments where each physical node can host multiple virtual machines, with each hosting several distributed memory application processes (e.g., MPI tasks), depending on the application level topology.

Due to increased need of to improve the performance of virtualized servers, many microprocessor vendors like Intel and AMD are providing hardware support (Intel-VT and AMD *Pacifica*) for maintaining several hardware states for several virtual machines in the microprocessor itself. This would reduce the changes made to the guest operating system servicing a virtual machine to and at the same time reap the benefits of para-virtualization. A physical node hosts a fully loaded operating system which consists of device drivers for I/O devices and network interfaces. Such a fully loaded host operating system is often referred to as the Host Operating System or DOM 0. As described earlier, the VMM creates an abstraction of the hardware resources (like the CPU, the memory, the Network Card and the I/O devices) per virtual machine instance. Such a virtual machine instance is serviced by an Operating System which is often referred to as a Guest Operating System or a DOM U. The DOM U uses the device driver interface provided by DOM 0 to access the hardware abstraction provided by the VMM. The VMM is responsible for translating the per-domain system call to access the actual hardware through the hypercall interface.

3 Related Work

The Virt-manager [9] Project from Redhat has provided an infrastructure to build a Virtual Node Manager which provides the system administrator with a GUI to display the performance information of the virtual machines along with all of its domains running on a physical node under a VMM. It also provides a Virtual Machine configuration window through which the system administrator can configure and create new Virtual machines on a physical node. Their goal is to provide a per physical node administration tool which can help the system administrators to monitor the exiting virtual machines and create new ones depending on a certain configuration. In contrast, our efforts are concerned with the managing of a cluster of physical nodes, each hosting multiple virtual machines and providing an infrastructure that encapsulates VMM level details in the system management tasks like dynamic load balancing, node maintenance, and preemptive node failure from the system administrator. It also provides a framework for designing new load balancing algorithms and resilience policies for handling node failures based on a cluster's telemetry data.

The libvirt API [10] project is a step forward in unifying the interface to gather the performance metrics of a virtual machine running over the hypervisor layer of a VMM. It provides an API in C which hides the hypervisor layer abstraction from the tool builders who want to develop performance analysis tools and cluster managers over many hypervisor layers. Our work, in contrast, can be built on top of the libvirt API and target multiple VMMs like Xen, qemu, VMWARE etc. Currently, our framework works only with Xen powered clusters.

4 Overview of Xen Virtual Cluster Manager

XCM is a Virtual cluster wide resource manager for managing Guest Domains running as Virtual machines over physical nodes. The XCM is built using a client-server model where it runs as a client on a remote node called as the Monitoring Station. Each physical node runs a XCM daemon which gathers performance metrics for the virtual machines running on that physical node. The XCM daemon uses the hypervisor interface to gather the performance metrics. The XCM daemons connect to the XCM client on the Monitoring Station. Then, the XCM client gathers the performance metrics from physical nodes using this daemon and aggregates the cluster-wide information in its internal data structures. The XCM client is built using C++ over the wx-Widgets GUI toolkit. The XCM daemons are built using C over the virtual machine monitoring libraries built over the Xen hypercall interface.

4.1 XCM Client

The XCM client runs on the monitoring station. The XCM client connects with the XCM daemons running on the physical nodes. The XCM client gathers the performance metrics from the XCM daemons at regular intervals and arranges them into a per-physical node per virtual-machine information. The time intervals at which the client gathers the results from the daemons can be configured by the user. The internal data structures of the client are used to display the performance of virtual machines on a per-physical node granularity in two views: the summary view and the detailed view.

The system administrator using the framework can study the performance metrics to make intelligent decisions about what administrative tasks need to be performed for optimal utilization of the physical resources. The administrator is also given an option of performing those tasks via the framework. Currently, three types of administrative actions can be performed by the administrator. These are dynamic load balancing, preemptive node maintenance, and live migration of individual virtual machines (henceforth referred to as domains) across physical nodes (henceforth referred to as nodes).

Views. XCM client can display virtual cluster-wide information in two views. The summary view displays coarse-grained information about the virtual machines running on each of the virtual nodes in a cluster. The detailed view displays the fine-grained per virtual machine/per virtual node information for all the virtual machines in a cluster.

Actions. The XCM client provides the system administrator with three types of actions that can be performed for managing the virtual cluster: dynamic load balancing, preemptive node maintenance, and live migration of individual domains. First, the XCM client can perform live migration of domains across virtual nodes. The information required to perform live migration is entered through the GUI interface. Second, the XCM client can perform automatic load balancing by migrating domains across all available physical nodes. Currently, we are investigating the policies on which we will determine our migratory decisions. Finally, the XCM client can also enable the system administrator enables the clean shutdown of a physical node by migrating all it's domains to other physical nodes in the cluster when that node is being taken away for maintenance.

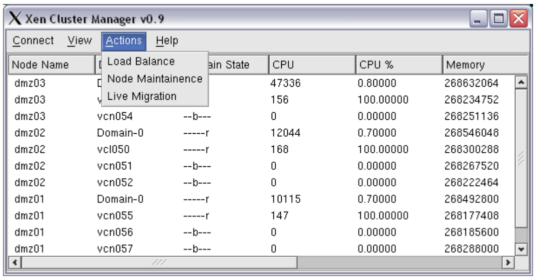
4.2 XCM Daemon

The XCM daemon runs on the physical nodes' host operating system. The daemon gathers performance metrics of all the virtual machines using the hypercall interface of the VMM. Currently, we gather per-VM performance metrics like total number of domains hosted by the VM, the state of each domain, the memory that has been allocated to each domain, the current CPU utilization of a VM, information about the number of virtual CPUs allocated to a VM, the number of virtual block devices and the information about the virtual network interface.

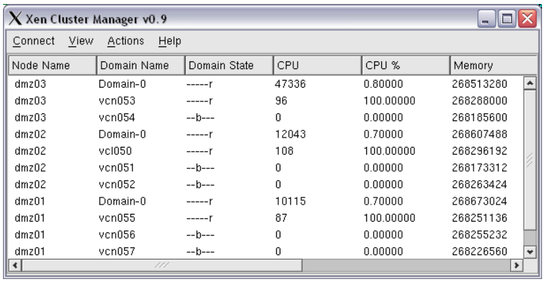
This information is obtained at regular intervals from the Xentop utility that is shipped along with the Xen distribution. The Xentop utility is built over the Xenstat library that provides an API for accessing the above mentioned performance metrics from the hypercall interface of the Xen VMM. The daemon is invoked using the command line interface and it connects to the monitoring station using TCP/IP sockets. The time interval that elapses between two successive performance data collect operations can be configured by the administrator using command line options. The daemon communicates the local performance data to the XCM client which organizes that data into its local data structures and displays that information on its GUI.

5 Xen Cluster Manager in Action

The XCM client provides the System Administrator of a virtual cluster with a capability to perform various administrative tasks like automatic load balancing, clean shutdown of nodes while system maintenance and live migration of individual domains from one node to the other across the virtual cluster. We have utilized the XCM client to manage a Virtual Cluster consisting of 3 physical nodes and hosting 8 virtual machines in all. The physical nodes are dual core AMD opteron machines running XEN-3.0.3 as a VMM. We have performed system administration tasks like live migration of domains across the three physical nodes, automatic load balancing of the virtual cluster based on a simple load balancing strategy, and clean removal of physical nodes during system maintenance. These experiments have been done as a proof of concept of our framework and demonstrate the scalability of our framework. The policies for various administrative actions are discussed next.



Node Name	D	ain State	CPU	CPU %	Memory
dmz03	D		47336	0.80000	268632064
dmz03	v		156	100.00000	268234752
dmz03	vcn054	--b---	0	0.00000	268251136
dmz02	Domain-0	-----r	12044	0.70000	268546048
dmz02	vcl050	-----r	168	100.00000	268300288
dmz02	vcn051	--b---	0	0.00000	268267520
dmz02	vcn052	--b---	0	0.00000	268222464
dmz01	Domain-0	-----r	10115	0.70000	268492800
dmz01	vcn055	-----r	147	100.00000	268177408
dmz01	vcn056	--b---	0	0.00000	268185600
dmz01	vcn057	--b---	0	0.00000	268288000



Node Name	Domain Name	Domain State	CPU	CPU %	Memory
dmz03	Domain-0	-----r	47336	0.80000	268513280
dmz03	vcn053	-----r	96	100.00000	268288000
dmz03	vcn054	--b---	0	0.00000	268185600
dmz02	Domain-0	-----r	12043	0.70000	268607488
dmz02	vcl050	-----r	108	100.00000	268296192
dmz02	vcn051	--b---	0	0.00000	268173312
dmz02	vcn052	--b---	0	0.00000	268263424
dmz01	Domain-0	-----r	10115	0.70000	268673024
dmz01	vcn055	-----r	87	100.00000	268251136
dmz01	vcn056	--b---	0	0.00000	268255232
dmz01	vcn057	--b---	0	0.00000	268226560

Fig. 1. XCM client in the Detailed view and also depicting administrative actions menu

5.1 Live Migration

Many times an administrator might want to remove a particular domain from a physical node and transfer it to run on some other physical node based on a certain performance metric. For example, a node may be hosting too many CPU intensive domains. This might reduce the throughput of a certain critical user-level application running on that domain. At the same time, there might be some node that might be only hosting a less CPU-centric application (e.g., an MPI application doing ping-pong communication). The XCM client gives the administrator a migration functionality in which they can select a source node, a source domain, and a new destination node for a domain. The live migration occurs transparently to the user-level application. Also, due to para-virtualization and hardware state saving of a virtual machine the overhead due to live migration is about 60ms.

5.2 Automatic Load Balancing

Sometimes, manual migration of virtual machines to across multiple nodes is a cumbersome task if the virtual cluster consists of a large number of physical nodes or virtual machines. Also, due to the arbitrary scheduling of virtual machines to host user-level applications, there might be inefficient usage of hardware resources. To ensure optimal hardware resource utilization, the administrator might develop some policies and algorithms which perform the load balancing of the virtual cluster by redistributing the cluster workload across multiple physical nodes. This typically requires the live migration of hundreds of domains across different physical nodes in the virtual cluster.

The XCM client enables the system administrator to perform automatic dynamic load balancing. Currently, the client uses a very simple resource management policy to define workload imbalance on a certain physical node. The initial load balancing policy seeks to balance the number of domains across the available physical nodes. We are currently extending our XCM framework to allow users to input specific policies, composed of the information provided by XCM. In this way, the administrator will be able to configure these policies based on weights given to various performance metrics collected by the XCM daemons. The current load balancing policy is based on the total number of domains in the cluster and the number of nodes in the cluster. Now, consider these three administrator defined metrics: Let N be the total number of physical nodes and D be the total number of domains in the virtual cluster. Also, let X be the number of maximum number of domains per physical node. Now, according to this policy, X will be evaluated by the expression: $X = (D / N) + 1$.

This policy limits the number of domains that can be hosted by a single physical node. We reiterate that this simple policy has been adopted to demonstrate this capability of our framework and may not be employed by real world installations of our framework. Now, whenever the system administrator selects the automatic load balancing menu item from the client's Actions menu, the XCM client parses through its internal data structures and makes a list of *migration information* (a tuple having *{source node, target domain, destination node}* information). Any node which contains more than D domains is selected as a source node and the nodes which have less than D domains are selected as the destination node. The target domains are randomly chosen from amongst the domains hosted on the source nodes. The XCM client maintains a migration information dispatch queue which stores all the pending live migration requests which have been requested by the system administrator. The "*Domain Migration Thread*" as described in section 5.1 clears this dispatch queue and sends the migration requests to the XCM daemons running on the source nodes. Hence, this dynamic load balancing strategy is application-agnostic and helps system administrator to devise load balancing policies in a virtual cluster.

5.3 Node Maintenance

Frequently, system administrators must service a physical node to upgrade and/or replace its hardware or software. Also, sometimes by monitoring metrics like heat generated per physical node, an administrator can make wise policies that can predict node failure in advance. Both these scenarios would require the administrator to remove a physical node from the virtual cluster. To make a clean removal of physical nodes from virtual clusters, the virtual machines hosted by the target physical node must be moved to a safe candidate node. Live migration can be used to evacuate a physical node of its VMs. The XCM client provides this feature to the administrator. The administrator chooses a particular physical node to be removed from the virtual cluster. The XCM client then rearranges the workload on a target node to be migrated to a safe destination node by filling out entries in the migration dispatch queue. Currently, a very simple algorithm is used to design this policy. The XCM client chooses the physical node which is hosting the minimum number of virtual machines as the destination node for the target node's domains. After these domains are migrated from the target node, the target node is proclaimed dead by the XCM client. This information is transmitted to the XCM daemon which causes it to terminate. The node can be safely switched off from the virtual cluster at that moment. Eventually, the dead node collector thread in the XCM client removes all the data structures related to the target node from its internal cluster-wide data structures.

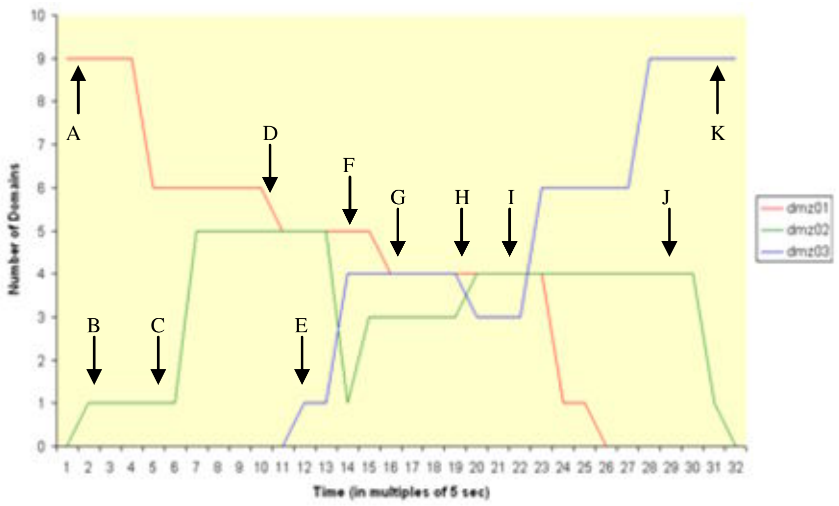


Fig. 2. Activity timeline for Xen migration experimental results

6 XCM Experiment

Using the XCM framework, we conducted an experiment comprising a variety of system administrative tasks on a virtual cluster configured to run 8 virtual machines across 3 physical nodes. Each physical node is a dual core AMD opteron machine running Red Hat Linux.

Each physical node hosts a Xen 3.0.3 VMM. On each of the 3 physical nodes, a patched fedora core 5 image was used to run as the host domain. The guest domains were also patched fedora core 5 images. In our experiments, the images were shared over NFS fileserver accessible to both the nodes. The nodes were connected through a gigabit Ethernet. The virtual machines running on these physical nodes are configured to run Linux fedora core 5 images.

Figure 2 represents a graph showing the virtual cluster workload distribution as a function of the number of domains on each of the three physical nodes as a function of time. We start of out experiment by starting a physical node and launching 8 DOM Us running on that virtual machine represented by "dmz1". At point A in the figure, there are total 9 domains running on dmz01. At time instance represented by pint B, a new physical node "dmz02" is added to the cluster. This physical node has only one domain (DOM 0) running on it. At point C, the administrator decides to balance the workload of the virtual cluster which causes the migration of 4 out of the 8 DOM Us to the newly available dmz02. The load balancing finishes at point D in time. Then at some point in time E, a new physical node, dmz03, is added to the virtual cluster. At point F, the administrator again decides to balance the cluster workload due to the availability of a free physical node. At point G in time, the virtual cluster is fully load balanced with each physical node hosting virtual machines. According to our load balancing policy discussed in section 5.2, the resources of the virtual cluster are being optimally utilized at this point in time. At time H, one domain is migrated from node dmz03 to the node dmz02 based on the cluster-wide performance metrics displayed by the XCM client. At point I in time, the administrator decides to schedule node dmz01 for system maintenance. All the domains of dmz01 are migrated to dmz03 depending on our policy as described in section 5.3. At point J in time, the node dmz03 is hosting 5 DOM Us. Then at point J in time, the administrator decides to remove node dmz02 as he decides to change the RAM configuration in that node. At point K in time, the virtual cluster only consists of one physical node, dmz03, which is now hosting all 8 DOM Us along with its DOM 0.

7 Conclusion

By providing a user-friendly framework built on top of TCP/IP sockets and the Xen hypercall interface for virtual cluster management we relieve the system administrator of a virtual cluster facility of manual interaction with individual virtual machines for performing administrative tasks like cluster performance monitoring, cluster load balancing, node maintenance, node failure etc. We also provide the administrator with a framework which can be used to automatically perform administrative actions by configurable policies designed using a variety of algorithms. This would reduce the burden of handling and maintaining a virtual cluster and would enable rapid decision making for optimal usage of hardware resources in such an environment.

References

[1] Bar, M.: Xen, the virtual machine monitor. Free Software Magazine, issue (June 5, 2005), `http://www.free/software/magazine/articles/focus-xen`

[2] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles SOSP 2003, Bolton Landing, NY, USA, October 19 - 22, pp. 164–177. ACM Press, New York (2003), `http://doi.acm.org/10.1145/945445.945462`

[3] `http://fabrice.bellard.free.fr/qemu/`

[4] Devine, S., Bugnion, E., Rosenblum, M.: Virtualization system including a virtual machine monitor for a computer with a segmented architecture (VMWARE). US Patent Office, Ed., USA (1998)

[5] `http://user-mode-linux.sourceforge.net/`

[6] Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live Migration of Virtual Machines. In: Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA (May 2005)
[7] Werner Fischer and Christoph Mitasch. High availability clustering of virtual machines – possibilities and pitfalls. Paper for the talk at the 12th Linuxtag, May 3rd-6th, Wiesbaden/Germany Version 1.01 (2006)
[8] Youseff, L., Wolski, R., Gorda, B., Krintz, C.: Paravirtualization for HPC systems. UCSB Computer Science Technical Report Number (2006)-10
[9] `http://virt-manager.et.redhat.com/`
[10] `http://libvirt.org/`

Deploying and Managing Xen Sites with XSM[⋆]

Felipe Franciosi[1], Jean Paulo Orengo[1], Mauro Storch[1], Felipe Grazziotin[1], Tiago Ferreto[2], and César De Rose[2]

[1] PUCRS/HP Research Center
Porto Alegre, Brazil
[2] Catholic University of Rio Grande do Sul
Computer Science Department
Porto Alegre, Brazil

Abstract. Xen is becoming a de facto solution for virtualization with low performance overhead. Virtualization features include optimization in resources utilization, server consolidation, improved security and fault tolerance. Server consolidation is currently one of the main attractions of virtualization for enterprises. It enhances the scalability of a regular IT infrastructure, enabling the utilization of an amount of servers larger than the available physical machines. However, deploying and managing virtual machines in such environment can be a complex task. It is first necessary to install the virtualization layer, represented by the VMM (Virtual Machine Monitor), on each physical machine. After that, the virtual machines can be deployed. Due to the possibility of having several virtual machines inside a single physical machine, an efficient management of the environment in order to enhance overall system performance and resources utilization is required. Considering these issues, we present in this paper the Xen Site Manager (XSM), a tool to facilitate the deployment and management of virtual sites based on Xen. It interacts with standard services and tools, such as SystemImager, DHCP, PXE and Ganglia, in order to provide high flexibility. We provide a detailed description of XSM architecture and present a performance evaluation of its deployment feature.

1 Introduction

Virtualization is one of the latest trends in the IT market. It's becoming a common standard due to the increase of commercial and open-source software products (e.g., Xen [1], VMWare [2], OpenVZ [3], Virtuozzo [4]) and investments of major hardware companies in providing mechanisms in their newest products to give better support to virtualization (e.g., Intel VT [5], AMD Virtualization [6]). Despite being an old concept (used initially by IBM 370 mainframes [7]), virtualization is proving to be an efficient solution to current problems faced by enterprises. Some of these problems are: exponential increase in the number of

[⋆] This work was developed in collaboration with HP Brazil R&D.

resources owned by organizations, high management costs, inefficient utilization of resources, etc.

One of the sectors presenting the higher increase in the utilization of virtualization techniques is data centers. Typical data center problems fit well in the types of problems that virtualization intends to solve. Usually, there is a huge number of services being executed on sub-utilized resources, which results in high management costs. In order to obtain an efficient utilization of all available computing infrastructure and provide good performance to all services, virtualization techniques must be applied.

In spite of virtualization seems to be a de facto solution to data center problems, its utilization is not straightforward, especially in environments with a huge number of resources. Deploying and managing a virtualization solution is currently a problematic task due to the nonexistence of flexible tools that provide a complete creation of a manageable virtualization environment. Current tools, such as Oscar [8], XenMan [9], and Enomalism Virtualized Management Console (VMC) [10] do not provide all the necessary features, from the installation of the virtualization layer in each resource, to the creation, removal, migration and monitoring of virtual machines in the environment.

Due to the lack of a flexible tool to deploy virtualization sites, we developed the Xen Site Manager (XSM). XSM is a tool to deploy and manage virtual environments. It is based on the Xen Virtual Machine Monitor [1], an open-source solution for virtualization. Xen is becoming a standard solution for virtualization due its low performance overhead to execute virtual machines. XSM is highly flexible in deploying virtual machines with different configurations in a regular site. It uses standard services and tools such as DHCP (Dynamic Host Configuration Protocol), PXE (Preboot Execution Environment), SystemImager [11] and Ganglia [12].

The remainder of this paper is organized as follows. Section 2 describes the XSM tool presenting all of its features. Section 3 presents a performance evaluation of XSM showing the time it takes to deploy variable configurations of Xen Sites. Section 4 presents some related work. Finally, Section 5 concludes the paper and presents some future works.

2 Xen Site Manager

When required to install a set of Xen servers in a data center, it is natural that a site administrator will look for tools to assist him with the job. Besides the help needed to install the machines, assistance to manage the virtual hosts is also required. This was the main motivation to develop XSM, a tool for deploying and managing Xen virtual machines in a site.

In order to fulfill those tasks, XSM uses available tools such as: SystemImager [11], Ganglia [12] and XM (Xen Management User Interface). While SystemImager and Ganglia are standard solutions for deploying and monitoring systems respectively, XM is a tool that comes along with Xen itself and is used to manage virtual machines. Figure 1 illustrates the typical lifecycle of XSM utilization, showing the concept of deploying the site, managing virtual machines and monitoring the environment. At any time, management of the previously installed hosts or deployment of new machines can be performed.

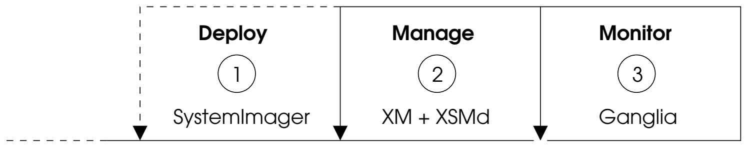


Fig. 1. XSM lifecycle

Jointly with the tools, XSM interacts also with another component, called Xen Site Manager Daemon (XSMd). This component, written in C language, was developed in order to perform management tasks in all hosts. It is based on a distributed management strategy. Each host contains a description of the whole site configuration. All XSMd communicate with each other in order to maintain a common vision of the site. The site can be managed from any host. When the site configuration changes, the configuration is modified in all other hosts. When the host receives the new configuration, it verifies if there is any local modification to be performed, and acts accordingly. This model excuses the need of a centralized management server, once any host may assume this role. Besides that, if any host goes down, the site will be manageable through any other host in the site.

2.1 Deployment

When installing more than one computer with the same system, automatic deployment becomes an attractive choice if compared to the manual installation of each host. XSM facilitates this process by using SystemImager and the PXE facility available in most computers. Figure 2 illustrates the boot process of a new computer involving the installation of the base system.

As the figure suggests, at least one auxiliary server is required for the initial deployment process. This server must run DHCP, TFTP (Trivial File Transfer Protocol) and SystemImager services. The first step consists of the new computer booting and, using PXE, fetching its configuration from the DHCP server. This configuration specifies, besides basic network configuration, the IP address of the TFTP server and the filename of a PXELinux image, which is downloaded in the second step. Next, this image is executed and the SystemImager client is downloaded. Step four consists of SystemImager client connecting to the server and fetching the base system, which is copied to the disk using partitioning information also specified by the server.

The base image supplied by XSM consists of a Linux Debian Etch as Domain 0. Naturally, this installation already ships a pre-configured Xen, Ganglia, XSMd

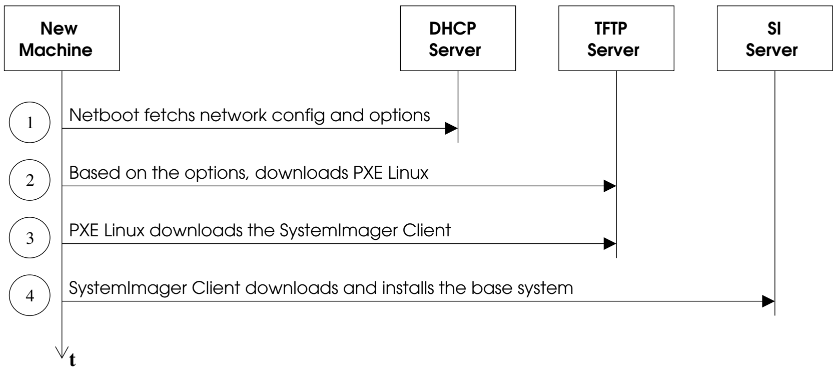

New Machine
DHCP Server
TFTP Server
SI Server
1
Netboot fetchs network config and options
2
Based on the options, downloads PXE Linux
3
PXE Linux downloads the SystemImager Client
4
SystemImager Client downloads and installs the base system
t

Fig. 2. Boot process using XSM deployment tools

and other Linux standard tools. As for the host configuration, SystemImager is also capable of automatically setting network interfaces (using DHCP or static setup) and hostname parameters.

As for the installation methods, SystemImager can use three different methods: unicast, multicast and broadcast. The choice depends on the site situation. With unicast, the SystemImager client will use the rsync tool to fetch the base image from the server. This method is well suited for a small number of hosts. When installing a larger number of hosts, the other methods, multicast and broadcast, are preferred due to the increase in network overhead with unicast. Deployment process are independent for each site machine. Thus, a failure in a host will not stop the deployment in the other site machines.

It is worth noting that, as all the tools used by XSM are administration tools, only the system administrator will be able to deploy a Xen site using XSM. No extra access control, but the one provided by the Linux System, is used during deployment process.

2.2 Management

After completing the deployment process, it is possible to manage the Xen site. XSM is capable of starting, stopping and migrating Xen VMs via a site configuration described in an XML file supplied by the user. An example of the site configuration file is presented in Figure 3. The example shows a site composed by 4 hosts, with one virtual machine per host (vm1, vm2, vm3, vm4). Virtual machines vm1, vm2 and vm3 are composed by Apache servers, and vm4 is a MySQL server. Hosts are identified by the IP address in the *DOM0* parameter. The parameters to configure a virtual machine are: virtual machine name, virtual machine configuration (kernel and image details), location (Domain 0 IP address), number of virtual CPUs, MAC address, method to obtain network configuration (static or DHCP) and physical memory. The virtual machine configuration represents the name of a regular Xen virtual machine configuration file which contains details to execute the virtual machine, such as: kernel, image, disks configuration, etc. Figure 4 presents the environment after the processing of the configuration file by the XSMd in each host.

```
<XSMConfig>
 <Set name="My_site">
  <Owner>root@domain.com</Owner>
  <VM name="vm1" config="apache_svr">
   <DOM0>192.168.5.101</DOM0>
   <VCPUS>4</VCPUS>
   <HWADDR>AA:00:00:00:00:01</HWADDR>
   <IP>DHCP</IP>
   <MEM>96</MEM>
  </VM>
  <VM name="vm2" config="apache_svr">
   <DOM0>192.168.5.102</DOM0>
   <VCPUS>4</VCPUS>
   <HWADDR>AA:00:00:00:00:02</HWADDR>
   <IP>DHCP</IP>
   <MEM>96</MEM>
  </VM>
  <VM name="vm3" config="apache_svr">
   <DOM0>192.168.5.103</DOM0>
   <VCPUS>4</VCPUS>
   <HWADDR>AA:00:00:00:00:03</HWADDR>
   <IP>DHCP</IP>
   <MEM>96</MEM>
  </VM>
  <VM name="vm4" config="mysql_svr">
   <DOM0>192.168.5.104</DOM0>
   <VCPUS>4</VCPUS>
   <HWADDR>AA:00:00:00:00:04</HWADDR>
   <IP>DHCP</IP>
   <MEM>96</MEM>
  </VM>
 </Set>
</XSMConfig>
```

Fig. 3. XSM site configuration file

When the site configuration file is modified, the XSMd reads the new configuration and forwards to other hosts. Each host reads the new configuration and performs the necessary modifications. XSMd interacts with XM in order to create, remove, reconfigure and migrate virtual machines. Virtual machines creation is performed when inserting a new virtual machine entry in the file. The

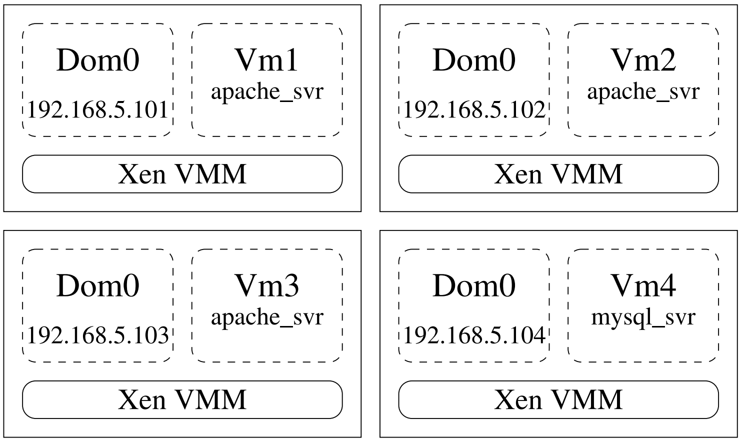


Fig. 4. Environment configuration

host that will contain the new virtual machine reads the file, creates a new Xen virtual machine configuration file with the generic configuration (virtual machine configuration) and the virtual machine parameters (VCPUs, HWADDR, IP and MEM), and starts the virtual machine. To remove a virtual machine, it's just necessary to delete the virtual machine entry in the file. Reconfiguration is performed modifying the values of the parameters, and migration changing the host (Domain 0) IP address of a virtual machine.

This distributed approach was chosen prior to a centralized management server in order to avoid a central point of failure and to simplify the development of the system according to an already existent infrastructure in our data center. Another advantage of such method is that no additional servers are needed for this purpose.

2.3 Monitoring

To keep track of all site state we use Ganglia, a scalable distributed monitoring system that was designed to monitor cluster and grid state [12]. On these environments, scalability, reliability and heterogeneity are common issues. To work with scalability, Ganglia was structured as a distributed system, with decentralized control. To detect a component failure, Ganglia uses a heartbeat protocol. It was also implemented and deployed over many different operating systems as Linux, FreeBSD, Solaris and IBM AIX.

Ganglia encompasses three main components: gmond, gmetad and gmetric. Each host of a XSM site runs gmond, which is responsible for collecting information about CPU usage, free memory, and other monitoring metrics. It also sends this information to a well-known multicast address every time significant updates occur. All hosts listen for metrics on the multicast address and collect and maintain monitoring data for all other hosts. Therefore, in case of failure, the view of the entire site state can be reconstructed easily because every host has an approximated view of the environment. In addition, Ganglia allows the user to easily expand the core metrics adding any arbitrary host metric using the gmetric component. On this work, we use gmetric to add information collected from the XM command about every VM including their current state and resource utilization.

Besides the common metrics included in Ganglia, the metrics collected for each virtual machine and Domain 0 are: CPU time and weight, maximum amount of memory and used memory, number of VCPUs (online and available), and status (blocked, ready, etc). Every metric is obtained through XM. Ganglia performs the periodic capture of these metrics and presentation with graphs showing a historical perspective.

3 Performance Evaluation

In order to evaluate the deployment performance, we built a testbed composed of 16 machines and one server. The machines share a common Fast-Ethernet Network (100Mbps). The server is responsible for executing basic services that provide support to the deployment system. This server is a Dual Pentium III 1 GHz, 512 MBytes RAM running Debian Sarge 2.6 and the following services: DHCP, TFTP, rsync, and SystemImager. The machines used as installation targets are all Intel Pentium 4 1.6 Ghz with 256 MBytes RAM and 20 GBytes of Hard Disk.

The experiment intends to analyze the performance of deploying a variable number of machines with our base image using two deployment techniques: unicast and multicast. The broadcast method was not measured because, to what is network concerned, its results would be the same as the multicast scenario. The size of our base image is 920 MBytes and, as described before, it consists of a Debian Etch Linux with Xen 3.0.1 and other software packages. It is also important to report that every measurement was executed 10 times, discarding the slowest and the fastest executions.

Figure 5 presents the results achieved. As expected, we obtained better results with the unicast method when using a small number of machines, once the installation is straightforward using TCP packets. The multicast installation uses UDP packets, which is unreliable and requires synchronization stages and error correction with retransmission, causing overhead for a single host installation. Nevertheless, this overhead is softened when installing multiple computers, because there's only one packet sent no matter the number of hosts involved in the process (considering no retransmissions).

This explanation helps our understanding of why the multicast curve looks logarithmic. That's because some computers may perform the installation faster than others, lagging the synchronization stages. With several similar computers in the deployment process, these stages are likely to be reduced, once the speed of the fastest and slowest computers is not very different.

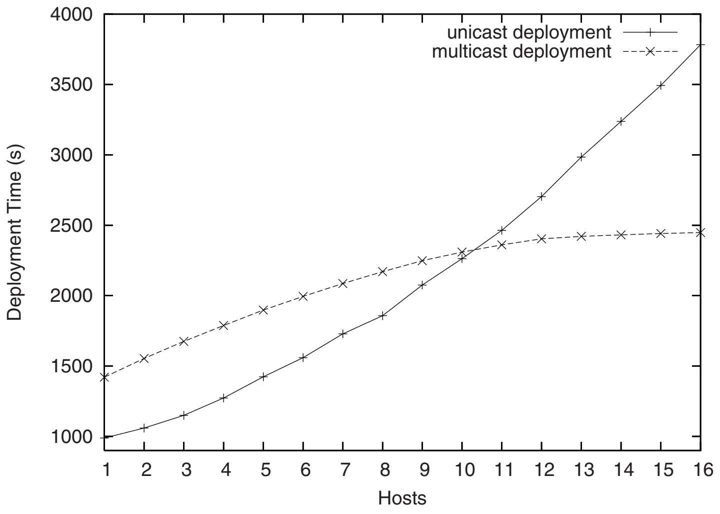


Fig. 5. XSM deployment performance evaluation

4 Related Work

The Xen project, although younger than its counterparts, was well received by the research community which elaborates many other projects to facilitate the deployment and management of Xen VMs, such as XenMan, Enomalism and Oscar.

The XenMan is an open source project that aims to provide an intuitive graphical user interface (GUI) for both administrators and new users of Xen. This interface, similar to the one provided by VMWare, can be used to create, start and stop VMs using buttons and fancy icons, making the management process easier. However, it was developed to work in a single computer, thus this tool is not feasible to manage one entire site.

Oscar, on the other hand, can be used to deploy a cluster composed of Xen VMs in one or more computer hosts. Oscar is a well-know project in distributed computing [13], used to assist the installation and management of Beowulf clusters. Recently, the Oscar suite was improved to allow the creation and deployment of Xen VMs, aiming to turn effortless their suite testing procedure [14]. Using this tool one can install Xen over a new machine, configure and create many VMs and also run them after the installation process. This tool can address the deployment steps of the XSM lifecycle presented in Figure 1; however, it cannot handle neither the management nor the monitoring steps.

A more complete tool that can manage and monitor Xen sites is the Enomalism Virtualized Management Console (VMC). The VMC provides a web page interface to monitor the VMs, showing the CPU and memory usage, among other information. With the same interface one can add new virtual machines or users and also manage them through a console emulator. To do these tasks the administrator need to be connected to the machine he wants to manage. In this way the VMC project is similar to XSM. However, the former is unable to deploy new machines and was built on a centralized point of view. The main advantages of XSM are that it assists the deployment steps and can handle failures because of the distributed approach used to manage the whole site. Moreover, the administrator can manage all the hosts by accessing only one of them rather than connecting to each host one by one, which is inadequate when there are many machines.

5 Conclusions and Future Work

In this paper we presented XSM, a tool to deploy and manage Xen Sites. XSM provides high flexibility in deploying virtual machines with different configurations in a regular site. It uses standard services and tools such as DHCP, PXE, SystemImager and Ganglia. We evaluated the deployment functionality of XSM and observed that it can both deploy a base Xen system efficiently in a site with a variable number of machines and start virtual machines described in a XML file.

As future work, we intend to enhance the management functionalities of XSM, develop a web-based interface to facilitate the management of Xen Virtual Machines, develop a tool for automatic generation of XML files, and perform deployment tests with larger environments in order to identify spots for optimization.

References

1. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauery, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of SOSP 2003 (2003)
2. Devine, S., Bugnion, E., Rosenblum, M.: Virtualization system including a virtual machine monitor for a computer with a segmented architecture. US Patent (October 1998)
3. OpenVZ Project: OpenVZ - Server Virtualization Open Source Project (2007), `http://openvz.org/`
4. SWsoft: Virtuozzo Server Virtualization (2007), `http://www.swsoft.com/en/products/virtuozzo/`
5. Intel Corporation: Intel Virtualization Technology (2006), `http://www.intel.com/technology/computing/vptech/`
6. Advanced Micro Systems: AMD's Virtualization Solutions (2006), `http://enterprise.amd.com/us-en/Solutions/Consolidation/virtualization.aspx`
7. Creasy, R.J.: The origin of the vm/370 time-sharing system. IBM Journal of Research and Development 25(5), 483–490 (1981)
8. Vallee, G., Scott, S.L.: OSCAR Testing with Xen. In: HPCS 2006: Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment, Washington, DC, USA, p. 43. IEEE Computer Society Press, Los Alamitos (2006)
9. XenMan Project: XenMan (2007), `http://sourceforge.net/projects/xenman/`

10. Enomaly Inc.: Enomalism Virtualized Management Console (2006), http://www.enomalism.com/
11. SystemImager Project: SystemImager (2007), http://systemimager.org
12. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: Design, implementation, and experience. Parallel Computing 30(7), 817–840 (2004)
13. Mugler, J., Naughton, T., Scott, S.L., Barrett, B., Lumsdaine, A., Squyres, J.M., t des Ligneris, B., Giraldeau, F., Leangsuksun, C.: OSCAR Clusters. In: Proceedings of the Ottawa Linux Symposium (OLS 2003), Ottawa, Canada (July 2003)
14. Vallée, G., Scott, S.L.: Xen-Oscar for cluster virtualization. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 487–498. Springer, Heidelberg (2006)

Xen Management with SmartFrog

On-Demand Supply of Heterogeneous, Synchronized Execution Environments

Xavier Gréhant[1,2], Olivier Pernet[3], Sverre Jarp[1], Isabelle Demeure[2], and Peter Toft[4]

[1] CERN openlab, Geneva, Switzerland
[2] École Nationale Supérieure des Télécommunications, Paris, France
[3] ENSIMAG, Grenoble, France
[4] Hewlett-Packard Laboratories, Bristol, UK
xag@cern.ch

Abstract. Applications to be executed on multipurpose Grids frequently have very specific resource requirements (platform, kernel, operating systems, libraries, memory, CPU, etc.) and need to be delegated part of the resource control. Typical Grid sites offer a limited range of resource types, inhibiting the range of applications that can be supported; and Grid node managers are bound to maintain their servers according to users requirements. To address these problems, we introduce SmartDomains, which combines the high performance virtual machine technology provided by Xen, with automatic deployment of Xen virtual machines using the SmartFrog configuration and deployment framework. SmartDomains automatically deploys distributed, synchronized pools of custom-configured Xen virtual machines and manages them through their lifecycle as a single coherent distributed execution environment. SmartDomains uses a representation of the complete distributed resources specifications, including information about how to sequence their creation and removal. We discuss SmartDomains test cases at CERN for distributed testbeds and Grid execution nodes.

1 Introduction

Although virtual machine (VM) technology has been around for four decades or more [1,2], there has been a resurgence of interest in recent years as virtualization has become practical on commodity hardware [3]. It follows that grid resource management tools will evolve to embrace support for virtual resources and those that do not will risk obsolescence. Our contribution to this evolution is a tool called SmartDomains (SD), which automatically supplies custom-configured, distributed virtual execution environments targeted at running batch Grid jobs or conducting system tests. In these contexts, it is important to keep independent the activity on the utilized resources and the maintenance of the backing hardware. SD provides fast, integrated VM control and a representation of resources to decouple administration and usage. A comparable *resource control plane* was developed in COD [4] for nodes provisioning in Beowulf clusters, and in PlanetLab [5] for managing networked applications' points of presence. Scientific production Grids (e.g. LCG, EGEE, OSG) do not yet optimise back-end resource usage with platform virtualization [6], which would add extra dimensions of flexibility in terms of resource configuration and fractional physical resource allocation (as illustrated with Tycoon [7]) although *Globus Virtual Workspaces* [8] already make interactions with the VM Monitors (VMMs) a Grid service.

The goal of SD is to provide a simple-to-use yet powerful mechanism for describing a required set of virtual machine resources, and a fully-automated deployment system to create VMs according to the supplied description. The automated deployment engine is a peer-to-peer distributed layer that takes in resource request descriptions and realises them with the requested sequencing. The same deployment engine is used to deactivate resource requests and release their resources, again according to the specified sequencing. Our experience with this approach leads us to believe that it makes it easier for resource administrators to prepare and control virtual resources, and for developers to create new functionality.

In this paper we discuss SD from three perspectives: In section 2 we explain the component technologies used in the SD system. SD usage is explained in section 3 along with a comparison of other systems using virtualization for resource management, and performance measurements are presented. In section 4 we illustrate the benefits of SD for resource administration, and its contributions to research and development in resource management systems.

2 SmartDomains, a Novel Approach

SD builds on Xen [9] for virtualization, and on SmartFrog [10,11] for the resource description and deployment mechanisms. Using virtualization in batch execution environments offers resource consumers and resource providers the following benefits:

Software compatibility: By creating a library of customized VM images, VMs can easily replicate a very wide range of resource configurations, satisfying the specific needs of a wide range of applications.

Resource sharing and performance isolation: By running multiple VMs on the same physical machine, fractional resources can be allocated, with fine-grained control over the resource consumption of each VM.

Failure isolation: VM failures do not affect the physical node nor other VMs.

We chose the Xen virtualization technology [9] for its high performance, openness, advanced features (live VM migration, for example) and growing popularity. Xen allows VMs to run at near native speed, which is critical for high-performance computing applications. However, the SD approach could be applied to other virtualization technologies such as VMWare [12], or in-kernel virtualization approaches such as KVM [13]. The resource description and deployment mechanism in SD is built using SmartFrog (SF), a Java-based framework for the configuration, deployment and management of distributed software systems, developed by HP Labs. In the domain of utility computing, the SoftUDC project [14] illustrates the use of SF along with VMMs for centralized management features. SF encourages the separation of the functionality of a software component from its configuration details. This allows the development of configuration-driven systems, the behaviour of which, including deployment choices, can be determined by configuration data. SF provides:

– A rich description language to express the configuration of software components (pieces of software to be placed on distributed hosts), and to express their orchestration at run-time using various composition components.
– A deployment engine formed from a distributed, peer-to-peer network of SF daemons. The deployment engine interprets the description language, dispatches software component deployment and management actions to local or remote daemons, checks liveness and maintains dependencies and references for attribute lookup and remote method invocation.

A domain is a running Xen virtual machine. SD defines a set of SF components that configure, monitor, deploy and manage Xen virtual machines. A user submits a description of the distributed environment he requires, specifying with complete freedom the kernels, filesystem images (distribution, libraries), computing resources (memory, CPU, hard drive size, etc), and orchestration details (e.g., the deployment sequence). SD automatically deploys the description and manages the deployment process until the resources are no longer required. Direct access to the physical nodes or to the Xen VMMs is not necessary. SD has been used in production since early in its development to create distributed virtual testbeds for the task of integrating and certifying gLite, a major grid middleware software distribution. This caused SD to support fast deployment of complex distributed configurations. We are now experiencing its integration in EGEE production Grid. In this context, SF dynamically boots appropriate execution environments upon request from a VO[1], and thus improves the compromise between resource delegation to the user, and control by the node manager.

3 Usage: A Comparative Overview

SD requires users to create descriptions for their virtual resource pools. There are simple extension and composition mechanisms that make it easy to create a library of different virtual resource pools, and to share these amongst users. It is simple to deploy a description, to un-deploy it, and to do so repeatedly. We compare this approach with other tools.

3.1 Launching a Virtual Pool

To start a deployment and launch a virtual pool, users submit a description with a single command line, and use another command line to un-deploy (remove)

[1] Virtual Organization, a federation of Grid users.

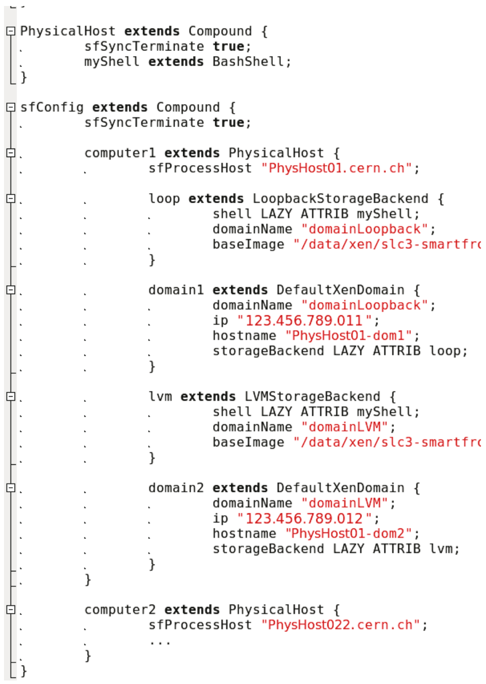
the virtual pool: `> sfStart localhost pool vp.sf`. The user specifies where deployment is to be initiated (*localhost* here), provides a name for identification at runtime (*pool*), and provides the description (*vp.sf*) of resource requirements: `> sfTerminate localhost pool`. Behavior on termination is defined in the description as well; by default it shuts down all virtual domains and cleans up the physical machines to return them to their initial state.

SD is distinctive in that it reduces the virtual resource management burden to the simplicity of an "on/off" button. It can automatically deploy the appropriate virtual resources on grid sites to handle incoming jobs or job workflows, or it can be used for repetitive, varied testbed setups for quality assurance tasks. A number of advanced enterprise resource management systems (Platform VM Orchestrator [15], Cassat Collage [16], OpenQRM [17], DynamicOE [18]) also leverage virtualization, but generally for a different purpose: they let the site administrator flexibly allocate computer center resources across long-lived applications, typically addressing resource utilization and high-availability concerns. Open source virtualization management projects (Virtual Workspaces, GPE, Enomalism) focus on presenting the VMM control via a variety of different interfaces. All these systems makes VMMs remotely accessible. SD development began with many of these ideas in mind, but its evolution was driven by specific requirements emerging from the CERN computing environment. Hence we focused on building a highly-configurable, highly-automated system that minimizes user interaction.

3.2 Describing a Virtual Pool

A single logical description specifies the actions that the distributed deployment system will take in order to deploy, and un-deploy, the requested virtual resource pool. The only current, known limitation to the run-time, distributed scope of a SD description is the presence of firewalls that block Java RMI communication, which is used for peer-to-peer communication by the SF daemons. A virtual resource pool deployment can vary in scale from one host to a complete data center. The RMI limitation will be overcome in future implementations by substituting RMI for a more firewall-friendly protocol, such as REST-based communication over HTTP. SD descriptions permit all possible configurations allowed by the Xen VMM, expressed as attribute values[2]. It differs in this respect from Amazon EC2, which provides fixed virtual machine resource configurations, and is elastic only in terms of the number of machines. However, like Xenoservers [19], EC2 does let users upload their own VM images. In Tycoon, it is the resource provider that decides what image is used. SD is linked to OSFarm[20] for custom image generation. In addition to configuring a domain, attributes define other behaviors, such as saving the image after use, compressing / uncompressing it. For ease of use, however, almost all attributes have default values. As a consequence, describing a single virtual domain with SD requires less knowledge than

[2] For brevity, they do not appear on figure 1. For a full list, please see the tutorial: `www.cern.ch/smartdomains`.



```
PhysicalHost extends Compound {
        sfSyncTerminate true;
        myShell extends BashShell;
}

sfConfig extends Compound {
        sfSyncTerminate true;

        computer1 extends PhysicalHost {
                sfProcessHost "PhysHost01.cern.ch";

                loop extends LoopbackStorageBackend {
                        shell LAZY ATTRIB myShell;
                        domainName "domainLoopback";
                        baseImage "/data/xen/slc3-smartfr
                }

                domain1 extends DefaultXenDomain {
                        domainName "domainLoopback";
                        ip "123.456.789.011";
                        hostname "PhysHost01-dom1";
                        storageBackend LAZY ATTRIB loop;
                }

                lvm extends LVMStorageBackend {
                        shell LAZY ATTRIB myShell;
                        domainName "domainLVM";
                        baseImage "/data/xen/slc3-smartfr
                }

                domain2 extends DefaultXenDomain {
                        domainName "domainLVM";
                        ip "123.456.789.012";
                        hostname "PhysHost01-dom2";
                        storageBackend LAZY ATTRIB lvm;
                }
        }

        computer2 extends PhysicalHost {
                sfProcessHost "PhysHost022.cern.ch";
                ...
        }
}
```

Fig. 1. Description with minimal configuration

writing a Xen configuration file. The description language lets the user configure how virtual machines should be synchronized (on figure 1 *Compound* is an example of synchronization type). It addresses the need for lifecycle management mechanisms cited on the Xen road-map [21], and is necessary for distributed batch jobs and tests. A management console provides a view of deployed virtual resource pools at runtime, and lets users update their configuration. For example, to change the frequency of virtual domains liveness checking, change memory allocation, or alter whether the image will be saved after shutdown.

3.3 Measurements and Future Work on Performance

Our measurements show that SD overhead is negligible on modern systems, for a considerable gain in resource flexibility. Our first tests with 48 CPU-intensive benchmark runs did not show any significant overhead introduced by SD. We found only a 0.25% difference in minimum elapsed times between these configurations: no liveness checking and liveness checking every 2 seconds, and between checks every 2 and 10 seconds. We monitored memory consumption in different scenarios (figure 2): a single deployment shows that an idle SD daemon needs about 20MB, and booting and terminating a first VM requires about 3 more MB. (We did identify a memory leak: each additional VM requires an additional 300KB which is never surrendered. For the moment the daemon can just

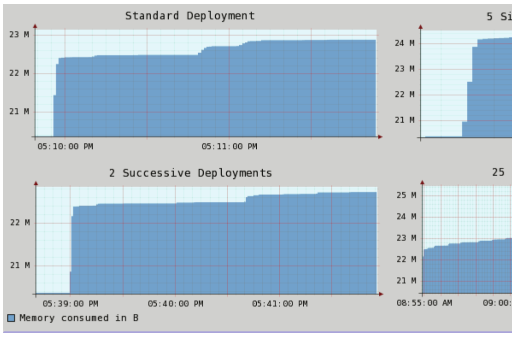


Fig. 2. Memory measurements

be restarted after every few hundred deployments.) For successive, cumulative deployments, we have been restricted by disk space so far. For simultaneous, multithreaded deployments, currently 8 VMs can be booted together on the same host; this limitation will be removed in the future. As shown on figure 2, booting 5 VMs simultaneously requires about 24 MB total, and all the VMs are booted after about 40 seconds. The time to change an environment is the same as VMPlants'[22] best case, a complementary work that minimizes software installation time on VMs.

4 Extending SmartDomains

SD is easy to extend for both resource administrators and developers. Administrators can easily provide and extend resource pool descriptions, that can be offered to their users. Developers can easily extend the SD framework through the addition of new control components.

4.1 Administration: Specialize by Composing and Pre-configuring

The first prerequisite is to install the Xen hypervisor on every physical machine. Installing SD is then just a matter of downloading the SD distribution and running an Ant command. It is therefore a simple process to install SD in large computer installations. And, while SD is targeted at large deployments, the simplicity of installation makes it quite usable on a single laptop for local VM deployments. Any computer can trigger the deployment of new virtual resource on the whole pool of physical hosts of which it is part, and where the virtual machines actually reside does not matter. There are no *a priori* privileged hosts and every node can act as the root for the whole site's computing power. SD distributes the work across the distributed computing environment and avoids bottlenecks and single points of failure. SD makes resource control simple in cases where it is delegated to a "trusted community" (e.g. physicists working on the same CERN experiment) or operated inside the same organization. In other cases, an additional admission control component may restrict deployment possibilities, and a user interface will regulate access to this component to restrict deployments. The extension mechanism (example of *domain1 extends XenDomainDefault* on figure 1) adds or overwrites component attributes that will be resolved at run-time as configuration data available to the corresponding class. This lets administrators create partly-configured descriptions which can be easily extended by users with all the necessary data. The description language allows attributes to be flexibly linked upwards (*ATTRIB*, *PARENT:*), or downwards (*name: childName: attribute*) in the hierarchy. Used in conjunction with LAZY keyword, this indicates a reference to be used for remote method invocation; without LAZY, it copies the target of the link. This allows administrators to define specialized components that populate low-level base components with user-level attributes. In our use-cases, these mechanisms improved convenience while using pre-defined templates.

4.2 Development: Enriching SD by Plugging in New Logic and Composite Structures

The basic SF approach is to describe a hierarchical organisation of software components and their configuration data; when deployed, each component interprets its configuration data, which drives the component's behavior. This proceeds in a hierarchy from the root component downwards. We mentioned how important it is for SD to not require any interaction with the user in order to accept batch requests for non-interactive jobs or software quality assurance tasks (section 3.1). Hence automation is vital, and new forms of automation can be developed by extending the set of SF/SD components. As an example, to dispatch virtual machines across physical hosts, a `Scheduler` component chooses the next suitable host, and a `Schedulee` wraps in the description the components to be placed.

For fine-grained resource sharing via a bidding system, a definitive advantage of Tycoon over previous works was the *best response algorithm* that bids in place of the user to avoids the need for frequent interaction. Indeed, we contend that computing resource management should be transparent and automated without restricting functionality. Combining the ability to manipulate descriptions with the ability to easily add new components allows us to balance functionality with automation and transparency. The mechanisms provided in SF for flexibly composing components at runtime allow developers to easily implement new behaviors (high availability, scheduling and load balancing mechanisms) and to apply these to the management of SD virtual resource pools. There is no restriction in the types of components that can be created, and they can easily fit into the framework and descriptions. Following the lessons of PlanetLab experience[5] SD favors evolutionary design more than *clean slate* design. In most other systems from enterprise resource management systems to research prototypes, adding new functionality can have significant implications for the whole system structure, thus restricted to some usage policies or available hardware (e.g. the two hard-coded availability classes in DynamicOE). SD allows the implementation of distributed algorithms involving VMs located on the whole peer-to-peer network of trusted daemons, to contrast with Tycoon where real-time auction for resources is limited to a competition between virtual machines on a per-physical-host basis [7].

5 Conclusion

SD deploys virtual resource pools for batch jobs or tests. Our use cases at CERN drove us towards high configurability, and high automation. Interaction is still possible, but the composable, component-based structure of the system allows automation functionality to be easily added. Generality is preserved when writing a description or specializing components, because the extension mechanisms allow descriptions to be prepared in advance and then customized for every deployment. Configurability, composition and lifecycle management are provided to the user or administrator through the description language. These are novel characteristics in a resource management system. they provide the necessary flexibility to decouple administration and usage requirements on the Grid *resource control plane*.

References

1. Goldberg, R.P.: Survey of virtual machine research. Computer, 34–45 (June 1974)
2. Denning, P.: Performance analysis: Experimental computer science at its best. Communications of the ACM 24(11), 725–727 (1981)
3. Rosenblum, M.: The reincarnation of virtual machines. ACM Queue 2(5), 34–40 (2004)
4. Chase, J.S., Irwin, D.E., Grit, L.E., Moore, J.D., Sprenkle, S.E.: Dynamic virtual clusters in a grid site manager. In: HPDC 2003: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, p. 90. IEEE Computer Society, Washington (2003)
5. Peterson, L.L., Roscoe, T.: The design principles of planetlab. Operating Systems Review 40(1), 11–16 (2006)
6. Nabrzyski, J., Schopf, J.M., Weglarz, J.: Grid Resource Management: State of the Art and Future Trends, 1st edn. Springer, Heidelberg (2003)
7. Feldman, M., Lai, K., Zhang, L.: A price-anticipating resource allocation mechanism for distributed shared clusters. In: EC 2005: Proceedings of the 6th ACM conference on Electronic commerce, pp. 127–136. ACM Press, New York (2005)
8. Foster, I., Freeman, T., Keahy, K., Scheftner, D., Sotomayer, B., Zhang, X.: Virtual clusters for grid communities. In: CCGRID 2006: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, pp. 513–520. IEEE Computer Society, Washington (2006)
9. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164–177. ACM Press, New York (2003)
10. Goldsack, P., Guijarro, J., Lain, A., Mecheneau, G., Murray, P., Toft, P.: Smartfrog: Configuration and automatic ignition of distributed applications. Technical report, HP (2003)

11. Pathak, J., Treadwell, J., Kumar, R., Vitale, P., Fraticelli, F.: A framework for dynamic resource management on the grid. Technical report, HPL (2005)
12. Waldspurger, C.A.: Memory resource management in vmware esx server. In: OSDI 2002: Proceedings of the 5th symposium on Operating systems design and implementation, pp. 181–194. ACM Press, New York (2002)
13. Qumranet: Kvm: Kernel-based virtualization driver. Qumranet White-Paper (2006)
14. Kallahalla, M., Uysal, M., Swaminathan, R., Lowell, D.E., Wray, M., Christian, T., Edwards, N., Dalton, C.I., Gittler, F.: Softudc: A software-based data center for utility computing. Computer 37(11), 38–46 (2004)
15. Computing, P.: Platform VM orchestrator., http://www.platform.com/
16. Cassatt: Cassatt collage., http://www.cassatt.com/prod_virtualization.htm
17. Qlusters Inc. 1841 Page Mill Road, G2, Palo Alto, CA 94304: openQRM Technical Overview: Open Source - Data Center Management Software (November 2006)
18. FusionDynamics: Fusiondynamics - http://www.fusiondynamics.com
19. 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 2004), San Francisco (December 2004)
20. Bjerke, H., Rattu, D., Habib, I.: OSFarm
21. XenSource: Xen roadmap, http://wiki.xensource.com/xenwiki/XenRoadMap
22. Krsul, I., Ganguly, A., Zhang, J., Fortes, J.A.B., Figueiredo, R.J.O.: Vmplants: Providing and managing virtual machine execution environments for grid computing. In: SC 2004: Proceedings of the ACM/IEEE SC 2004 Conference on High Performance Networking and Computing, Pittsburgh, PA, USA, p. 7. IEEE Computer Society, Los Alamitos (2004)

Integrating Xen with the Quattor Fabric Management System

Stephen Childs and Brian Coghlan

Department of Computer Science, Trinity College Dublin
childss@cs.tcd.ie

Abstract. While the deployment of virtual machines (VMs) within the high-performance computing (HPC) community is proceeding at a great pace, tools for system management of VMs are still lagging behind those available for physical machines. In order to make further progress, VM management must be fully integrated with existing fabric management infrastructure. We present the results of work done to integrate Xen [4] with the Quattor [15] fabric management suite. The principal contributions are the development of a network bootloader for para-virtualised Xen VMs and a Quattor management component for setting up hosted VMs. The combination of these tools allows for full unattended installation of Xen VMs and the automatic configuration of services, all from a single configuration database.

1 Introduction

Deployment of virtual machines (VMs) in the high-performance computing community is taking off rapidly. The past few years have seen the use of virtualisation for Grid service nodes [8], training infrastructures [5,6], dynamic clusters [11], and middleware certification. Initially most projects took an ad hoc approach to the creation and management of virtual machines, developing small tools as necessary. However, as virtualisation becomes mainstream and starts to be deployed in production environments, the need to integrate VMs into existing management infrastructure has become apparent: it should be possible to use existiing installation and configuration mechanisms to install virtual machines.

The term fabric management system is used to describe an integrated suite of tools used to install, configure and monitor the various service and compute nodes that make up a Grid or HPC infrastructure. Quattor [15] is a fabric management system originally developed as part of the European DataGrid [17] project, and now developed as a community effort with contributions from various sites, mostly within the high-energy physics community. It is currently deployed at over forty sites managing over 10,000 nodes. It is a modular system designed to manage all stages of a machine's lifecycle from initial installation on the bare metal to configuration (and reconfiguration) of complex software such as Grid middleware.

Quattor configurations are written in Pan [9], a powerful declarative language allowing complex compositions of configuration templates, and are compiled to

XML profiles that contain a complete configuration tree for the machine. Configuration profiles for a site are stored in a configuration database, and the client machines pull profiles from this database at regular intervals. Programs running on the client (known as components) read the relevant sections of the configuration profile and generate configurations for the services they manage. For example, the `ncm-accounts` component manages user accounts, and `ncm-ssh` manages secure shell (ssh) services.

Quattor also includes a subsystem for configuring a network boot infrastructure. The Automated Installation Infrastructure (AII) uses the information from machine profiles to generate configuration entries for DHCP and pxelinux [3] running on an installation server. AII allows machines to be installed from scratch using Kickstart[1]: once the server and client are configured correctly for network booting, installation of a new node is as simple as switching it on. The aim of this work is to fully integrate Xen virtual machines with AII and Quattor so that they can be automatically installed in the same way as physical machines.

2 Design

The aim of this work is to integrate VMs fully into the fabric management system so that the complete lifecycle can be managed automatically via the configuration database and installation servers. Once correct profiles have been configured for a VM and its host, it should be as easy to install the VM as it is to install a physical machine.

Integrating Xen with Quattor required three main areas of work: i) design of appropriate Quattor data structures to represent Xen VMs and global configuration options, ii) development of a Quattor management component for Xen (`ncm-xen`), and iii) development of a bootloader for para-virtualised Xen domains that was compatible with automatic network installation. While the first two items are specific to Quattor, the bootloader we have developed is useful for other systems that use network booting (especially those using pxelinux).

It should be noted that there was no need to modify core Quattor services. The aim was to make VMs behave as much like physical machines as possible, rather than adapting the core services to treat VMs specially. The combination of a configuration component running on the hosting machine and enhancements to the VM boot process made this possible. This approach results in a simpler system: the installation procedure for VMs is very similar to that of physical machines, and so the normal debugging procedures used by administrators apply. Also, once a VM has been successfully booted and configured, day-to-day management via Quattor is exactly the same as for a physical machine.

2.1 VM Configuration

In Quattor, the configuration state of a physical machine can be entirely encapsulated in its configuration profile. In contrast, there are two aspects to the configuration of a virtual machine: firstly, the configuration of the VM itself, and secondly the information needed by the host machine to set up the VM correctly. The first category includes the same information as for a physical machine (e.g. user accounts, service parameters, etc.) The second category includes all the information needed by the host machine to set up the virtual machine: location of storage, network configuration (including MAC address to be assigned), amount of memory to be assigned, method of booting, etc. None of this information is visible within the VM, and so must be included in the configuration of the host machine.

```
"/software/components/xen/domains" = push(nlist(
"memory", value("//cagnode50/hardware/ram/0/size"),
"name", "cagnode50.cs.tcd.ie",
"disk",list(nlist
           ("type",'file',
            "path",'/var/xen-grid/cagnode50/fs/disk',
            "device",'sda',
            "size",value("//cagnode50/hardware/harddisks/sda/capacity"),
            "rw",'w'),

           nlist("type",'lvm',
            "hostdevice",'xenvg',
            "hostvol", 'cagnode50-swap',
            "size", 6*GB,
            "device",'hda3',"rw",'w')),
"vif",list('mac='+value(//cagnode50/hardware/cards/nic/eth0/hwaddr)),
"bootloader","/usr/bin/pypxeboot",
"bootargs","vif[0]",
"auto", true
));
```

Fig. 1. Example domain configuration

In Quattor, the configuration for a particular component is normally located under the location `/software/components`. So the configuration for `ncm-xen` is located under `/software/components/xen`. The profile of a host machine includes a list (called `domains`) of data structures for the VMs that it hosts. Figure 1 shows an example of this data structure. Most of the entries are simple strings that will be directly translated to entries in the Xen configuration file. The exception is the `disk` entry, which is a more complex data structure incorporating information needed to create the physical filesystem (i.e. the size) as well as that needed for VM configuration. Much of the information needed to create the VM can be extracted from the VM's own profile (e.g. RAM size, disk capacity, MAC address). The `auto` flag determines whether `ncm-xen` will set up the links necessary to make the VM run automatically on host system startup.

In addition to the data structures representing domains, there are other variables that control the global configuration of Xen on the host, and the operation of `ncm-xen`. For example, `ncm-xen` can optionally create storage for guest VMs:

this is controlled by the `create_filesystems` variable. The `create_domains` variable determines whether VMs are automatically started when `ncm-xen` detects that they are not running.

When `ncm-xen` is run on the host machine, it extracts this list of data structures representing VMs from the machine's profile. `ncm-xen` uses this information to generate Xen configuration files for VMs and to set up storage. Once the VM has been installed successfully, it will retrieve its own profile and invoke the Quattor configuration components to configure services.

3 Implementation

3.1 Network Bootloader

Preboot Execution Environment (PXE) is a standard for automatically retrieving operating system kernels and configuration information over the network at boot time. It is widely used for installation of nodes within the high-performance computing community. When a machine configured for PXE boots up, its network card broadcasts a request for an IP address. A DHCP server will respond with an IP address and the address of the server holding the configuration and OS kernel for that machine. The PXE client server then downloads the OS kernel and uses it to boot the machine.

PXE booting is often used in conjunction with an automatic system installation program such as Kickstart [1]. Kickstart reads a configuration file, sets up filesystems and installs a basic set of packages to get the OS up and running.

PXE clients are usually located either in a boot ROM on the network interface or on the motherboard itself. Boot images loaded from a floppy disk or other bootable media are also sometimes used. This option would be possible for fully virtualised VMs running on machines with hardware support, but could not be used with the para-virtualised (PV) VMs that are currently widely deployed. As a PV Xen virtual machine does not have physical hardware, some other solution is required. Various approaches have been suggested in discussions on the Xen developer list: we have taken the simple approach of writing a program that runs on the host VM and performs a simulated PXE boot process on behalf of its guest VMs.

Xen supports the use of a "bootloader" for guest VMs. This is simply a program that is run on the host VM (as part of the VM boot process) to retrieve a kernel for a guest VM. This scheme has been used by others to implement `pygrub`, a program that looks inside the guest's filesystem, works out which kernel to use, and copies it back out to the host's filesystem to boot the VM. We have implemented a network bootloader, `pypxeboot`, that takes a similar approach but which retrieves the configuration and kernel over the network rather than from the guest's filesystem. The Trivial File Transfer (TFTP) [18] protocol is used for downloads.

`pypxeboot` is invoked as part of Xen's VM creation procedure, and performs the following steps (illustrated in Figure 2) to retrieve a boot kernel:

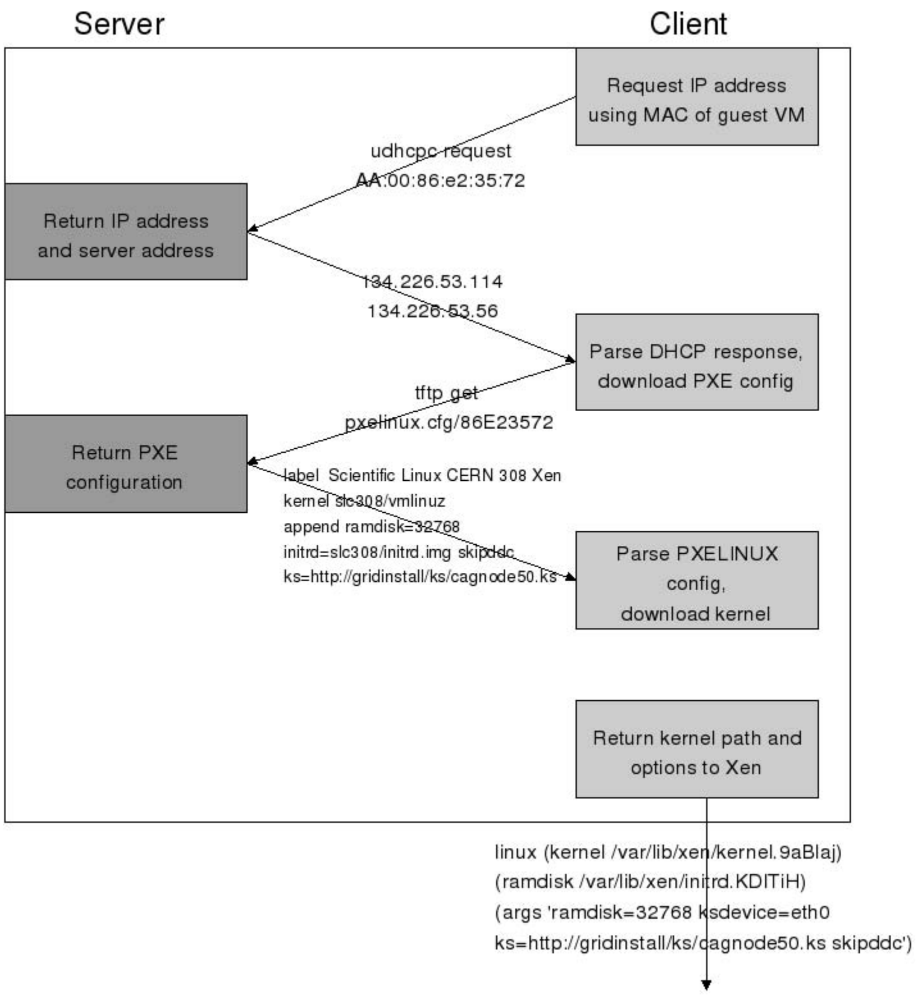

Server
Client
Request IP address using MAC of guest VM
udhcpc request AA:00:86:e2:35:72
Return IP address and server address
134.226.53.114
134.226.53.56
Parse DHCP response, download PXE config
tftp get pxelinux.cfg/86E23572
Return PXE configuration
label Scientific Linux CERN 308 Xen
kernel slc308/vmlinuz
append ramdisk=32768
initrd=slc308/initrd.img skipddc
ks=http://gridinstall/ks/cagnode50.ks
Parse PXELINUX config, download kernel
Return kernel path and options to Xen
linux (kernel /var/lib/xen/kernel.9aBlaj)
(ramdisk /var/lib/xen/initrd.KDITiH)
(args 'ramdisk=32768 ksdevice=eth0
ks=http://gridinstall/ks/cagnode50.ks skipddc')

Fig. 2. Pypxeboot network boot cycle

1. Make a DHCP request using the MAC address that will be assigned to the guest VM.
2. Extract the address of the TFTP server and download the configuration for the guest VM.
3. Read the configuration received. If the VM is set to network boot, then download the kernel for the VM over the network. If the VM is set to local boot, drop back to a bootloader that will load the kernel from the VM's own filesystem. In either case, the location of the kernel retrieved is passed to Xen, which uses it to boot the VM.

DHCP client software normally reads the MAC address from a physical interface and uses that directly. With `pypxeboot`, the host VM will need to make a request on behalf of the guest VM using the guest's MAC address. We modified the `udhcp` [10] client to read a user-specified MAC address from the command line, allowing us to send a DHCP request that appears to come from the guest VM. `udhcp` also retrieves other parameters provided by the server, the most important being the location of the server holding the node's boot configuration and kernel image. `pypxeboot` parses the output from `udhcp` to determine which server to use, and then downloads the configuration using a TFTP client.

In order for a Kickstart installation to succeed, installation tools such as Anaconda must be modified to run on the Xen virtual hardware. We are currently using images created by Linux Support at CERN [16] which include modified partitioning and hardware detection code that is compatible with Xen virtual hardware.

3.2 Quattor Management Component

We have implemented `ncm-xen`, a management component that runs on the host machine to translate the configuration specified in a machine's profile into a valid Xen configuration. This requires Xen global configuration options to be set, and also configuration files to be created for the guest VMs. `ncm-xen` can also create storage to be used by guest VMs and start VMs: these optional features are required to enable full unattended installation of Xen hosts and VMs from scratch.

Quattor components are normally written in Perl, and use the Configuration Cache Management (CCM) API to access the Quattor profile for the machine on which they are running. As shown above, most of the parameters for a VM can be translated directly into Xen configuration parameters: `ncm-xen` extracts these entries from the profile and writes them out to a configuration file for the VM.

A little more work is necessary to correctly configure the filesystems for a VM. In addition to creating the appropriate configuration entries describing the storage to be used by the VM, `ncm-xen` also supports the creation of file-backed storage and logical volumes. If a file or logical volume is specified in the VM's configuration entry but does not exist, `ncm-xen` will create it. This is particularly useful when host and guests are being installed from scratch: `ncm-xen` can automatically set up backing storage for a VM prior to its first instantiation. The details from the profile are also used to generate entries in the correct format for the Xen configuration file.

3.3 Putting It All Together

Having presented all the components of our integrated system, we can now describe a complete usage scenario where a combination of host and guest VMs, each with its own profile, are automatically installed on a physical machine.

Figure 3 shows the complete automatic installation of a host and guest VMs, from bare metal to a fully-configured set of VMs. The first step is to set up the DHCP entries and pxelinux configuration on the installation server: the AII

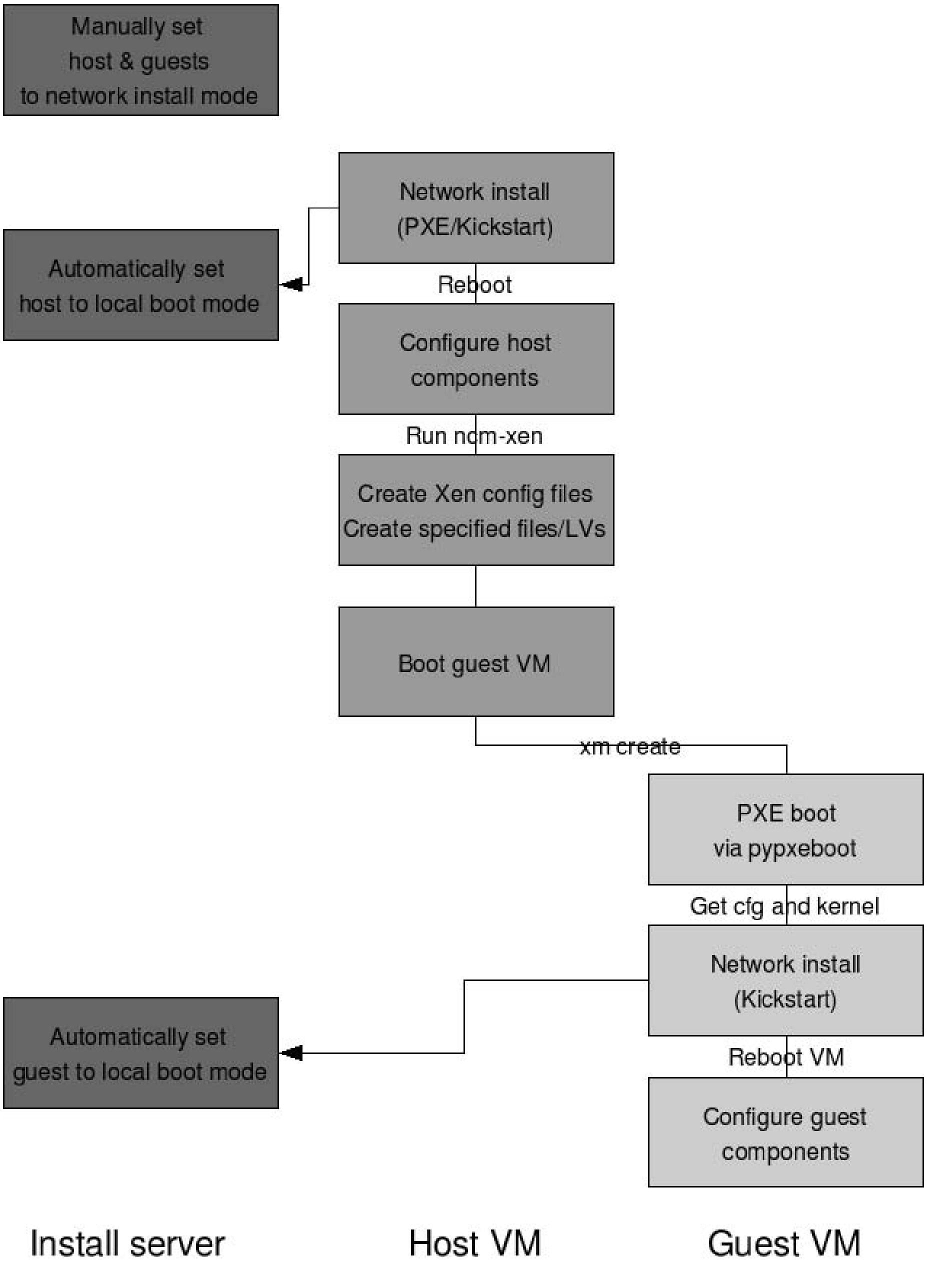

Manually set host & guests to network install mode
Network install (PXE/Kickstart)
Automatically set host to local boot mode
Reboot
Configure host components
Run ncm-xen
Create Xen config files
Create specified files/LVs
Boot guest VM
xm create
PXE boot via pypxeboot
Get cfg and kernel
Network install (Kickstart)
Automatically set guest to local boot mode
Reboot VM
Configure guest components
Install server
Host VM
Guest VM

Fig. 3. Complete installation cycle

program provides options to translate information from machine profiles into DHCP entries, and then to set up the pxelinux configuration for the machines to cause a network installation.

The next stage of the process starts when the physical machine is powered on for the first time. The machine then boots into Kickstart via PXE and installation starts. Once a minimal set of packages has been installed, the machine restarts and boots from local disk. At this stage, Quattor packages are installed, the machine's profile is downloaded and a configuration run begins. As part of this configuration run, `ncm-xen` will be invoked to configure guest VMs. It will read the entries from the host's profile and then create filesystems and Xen configuration files according to this information. The guest VMs are configured to use `pypxeboot` as their bootloader, and when they are started they determine which kernel to boot via PXE, then boot into the same Kickstart installation procedure used by the host. They then reboot, download their Quattor profile and begin a configuration run to set up their own services. Once this completes, the host and guest VMs are all under Quattor control and future configuration changes can be made by deploying new profiles.

4 Related Work

The need to integrate VMs with existing management systems has been addressed by others. The developers of OSCAR [12], have presented the results of a project [2] to add support for Xen guests. However, as they did not solve the problem of network booting para-virtualised Xen guests, they were forced to use a cumbersome solution involving a two-stage install. In Xen-OSCAR, the guest VM initially boots into an install image that sets up the real filesystem for the VM and installs into it. As our installation uses a native network boot, the process is much simpler. The installation process for a VM is also very similar to a standard network installation, making it easier to debug.

XenSource provide the XenEnterprise [14] management suite for deployment and control of Xen VMs. Enomaly's enomalism Virtualized Management Dashboard provides similar functionality [13]. Both of these products provide a GUI-driven interactive management methodology, rather than the fine-grained, automated control provided by the Pan language and configuration database in Quattor. We see the two modes of management as complementary: in fact, we are also developing an interactive management interface targeted at the creation of custom test environments. This tool, known as GridBuilder, [7] uses template FS images and copy-on-write techniques to rapidly create transient VMs whose configuration need not be stored permanently in a central database. We hope to investigate closer integration of GridBuilder and Quattor in the future: for example, experimental configurations created in GridBuilder could be translated into Quattor configurations for deployment in a production environment.

5 Conclusion

The integration of Xen with a fabric management suite overcomes a significant obstacle to further deployment of VMs within the HPC community. Previously, those wishing to deploy VMs needed to develop their own tools, or to interface external VM management tools to their existing fabric management system. The tools we have developed allow for seamless integration of VMs into an existing Quattor infrastructure, providing for unified management of physical and virtual machines.

`ncm-xen` is currently being used by Grid-Ireland to install Gird infrastructure servers hosted on VMs, and has been released to the Quattor community for testing. `pypxeboot` has been submitted for inclusion in the Xen release.

Future work will focus on extending the functionality of `ncm-xen`. For example, support for downloading and customising pre-installed images would allow for faster startup compared to from-scratch installation. It would also be good to modify the schema so that it is compatible with other VM technologies such as VMWare, qemu, and kvm.

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

References

1. Kickstart installations. `http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/ch-kickstart2.html`
2. Vallée, G., Scott, S.L.: Xen-Oscar for cluster virtualization. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 487–498. Springer, Heidelberg (2006)
3. Peter Anvin, H.: PXELINUX - SYSLINUX for network boot., `http://syslinux.zytor.com/pxe.php`
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, ACM, New York (2003)
5. Berlich, R., Hardt, M.: Grid in a box - virtualisation techniques in Grid training. In: EGEE conference, Athens (April 2005), Available via: `http://www.ep1.rub.de/~ruediger/pandoraAthens.pdf`
6. Cassidy, K., McCandless, J., Childs, S., Walsh, J., Coghlan, B., Dagger, D.: Combining a virtual grid testbed and grid elearning courseware. In: Proc. Cracow Grid Workshop 2006 (CGW 2006). Academic Computer Centre CYFRONET AGH, Cracow, Poland (October 2006)
7. Childs, S., Coghlan, B., McCandless, J.: GridBuilder: A tool for creating virtual Grid testbeds. In: 2nd IEEE Conference on eScience and Grid computing, Amsterdam (December 2006)
8. Childs, S., Coghlan, B., O'Callaghan, D., Quigley, G., Walsh, J.: A single-computer grid gateway using virtual machines. In: Proc. AINA 2005, Taiwan, March 2005, pp. 761–770. IEEE Computer Society, Los Alamitos (2005)
9. Cons, L., Poznanski, P.: Pan: A high-level configuration language. In: LISA 2002: Sixteenth Systems Administration Conference, Usenix, pp. 83–98 (2002)

10. Dill, R., Ramsay, M.: udhcp client/server package (2002), `http://udhcp.busybox.net/`
11. Emeneker, W., Stanzione, D., Jackson, D., Butikofer, J.: Dynamic virtual clustering with Xen and Moab. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 440–451. Springer, Heidelberg (2006)
12. Open Cluster Group. Open source cluster application resources., `http://oscar.openclustergroup.org`
13. Enomaly Inc. enomalism virtualized management dashboard., `http://www.enomalism.com`
14. XenSource Inc. Xen Enterprise 3.1 datasheet., `http://www.xensource.com/files/xenenterprise_3.1_datasheet.pdf`
15. Garcia Leiva, R., Barroso Lopez, M., Cancio Melia, G., Chardi Marco, B., Cons, L., Poznanski, P., Washbrook, A., Ferro, E., Holt, A.: Quattor: Tools and Techniques for the Configuration, Installation and Management of Large-Scale Grid Computing Fabrics. Journal of Grid Computing 2(4) (2004)
16. Jaroslaw Polok. Xenification of Scientific Linux CERN., `https://twiki.cern.ch/twiki/bin/view/LinuxSupport/XenificationOfSLC`
17. Segal, B., Robertson, L., Gagliardi, F., Carminati, F.: Grid computing: The european data grid project (2000)
18. Sollins, K.: The TFTP Protocol (Revision 2), RFC (1350) (July 1992)

Getting 10 Gb/s from Xen: Safe and Fast Device Access from Unprivileged Domains

Kieran Mansley, Greg Law, David Riddoch,
Guido Barzini, Neil Turton, and Steven Pope

Solarflare Communications, Inc.
{kmansley,glaw,driddoch,gbarzini,nturton,spope}@solarflare.com

Abstract. The networking performance available to Virtual Machines (VMs) can be low due to the inefficiencies of transferring network packets between the host domain and guests. This can limit the application-level performance of VMs on a 10 Gb/s network. To improve network performance, we have created a "virtualization aware" smart network adapter and modified Xen[1] to allow direct, but safe, access to such adapters from guest operating systems. Networking overheads are reduced considerably, and the host domain is removed as a bottleneck, resulting in significantly improved performance.

We describe our modifications to the Xen networking architecture that allow guest kernels direct — but secure — access to the networking hardware, whilst preserving support for migration. We also describe briefly how the same technology is used to grant direct network access to user-level applications and thus provide even greater efficiency in terms of bandwidth, latency and CPU utilisation.

1 Introduction

Modern commodity servers are able to saturate 10 Gb/s Ethernet networks [1]. However, VM guests incur significant additional overheads due to context-switching, data movement and passing packets between the host domain and guests. The Xen [2] virtual machine architecture imposes relatively low overheads on networking compared with other virtualization technologies, yet its performance is well below that of native operating systems [3, 4]. The privileged host VM quickly becomes a bottleneck as all network traffic must pass through it.

This I/O bottleneck makes virtualization effectively impractical for certain classes of application, including many HPC applications and high performance servers. As 10 Gb/s Ethernet moves more into the mainstream, the set of applications for which this is a barrier to adoption of virtualization will likely increase.

[1] Xen is a trademark of XenSource, Inc. in the United States and other countries.

This paper shows how allowing guest operating systems direct access to the I/O hardware eliminates the bottleneck and software overheads usually associated with virtualized I/O, and allows a virtualized guest to achieve performance comparable with a system running natively. Doing so requires additional support from the hardware in order to multiplex the device between multiple guests that may access it concurrently and also to enforce isolation so that guests cannot gain privileges or compromise system integrity.

The rest of this paper is structured as follows. Section 2 outlines the architecture for accelerated networking that we have added to Xen, while Section 3 expands to describe our implementation and the results we have obtained with a virtualization-aware network adapter. Section 4 shows how the same techniques can be applied to user-level applications. Finally, Section 5 concludes.

2 Architecture

2.1 Xen Paravirtualized Network I/O

Paravirtualized network I/O in Xen is achieved through a pair of interlinked drivers; *netfront* the "frontend driver" in the guest, and *netback* the "backend driver" in the host domain. The frontend and backend communicate through a region of shared memory and send each other virtual interrupts using event channels. Together these form a channel that supports the transfer of packets between host domain and guest.

The upper edge of the frontend driver presents the interface of a standard network device driver, allowing it to interface to the bottom of the guest's network stack. The backend appears likewise and is usually configured to connect to a software bridge in the host OS. This allows it to communicate with the host's network stack, other virtual machines' backend drivers, and physical network interfaces and so the network beyond.

Packets that arrive from a physical network interface are routed by the bridge to the appropriate backend drivers, which in turn forward them to the corresponding guests' frontend drivers. These then pass them on to the network stack as if they had arrived directly at the guests. Packets sent by guests follow the same path in reverse. Packets sent from one guest to another are routed between the corresponding backend drivers by the bridge.

2.2 Acceleration Architecture

We have extended the Xen netfront/netback architecture with a plugin interface that provides an opportunity to accelerate network performance. We have preserved the existing netfront/netback channel and added an optional "fast path" implemented by the plugins. To account for as many different kinds of hardware as possible (existing and future designs), the plugin interface makes as few assumptions as possible. By preserving the existing netfront/netback mechanism it is possible to support migration and also hardware that accelerates only a subset of traffic with other traffic taking the "slow path".

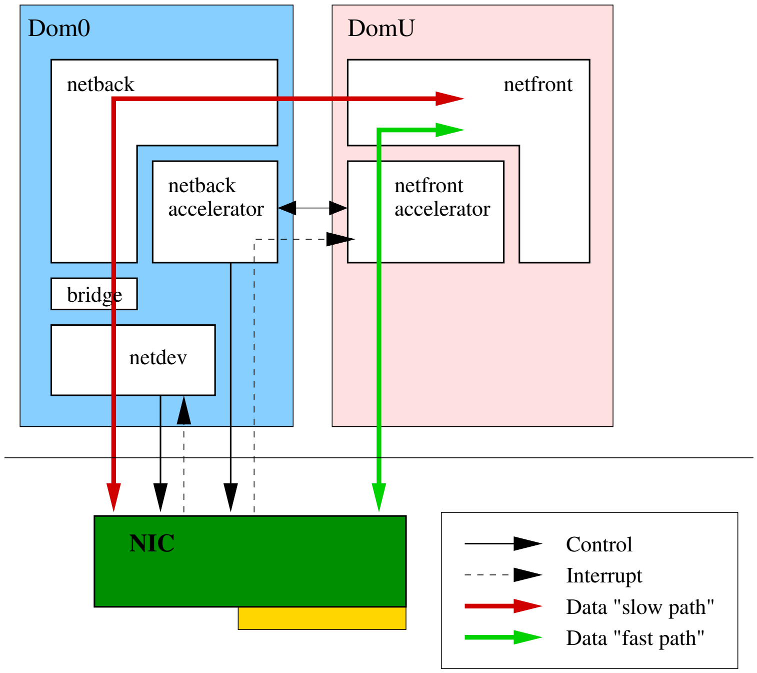


Fig. 1. Accelerated networking in Xen

Various acceleration techniques are possible, but the main model anticipated is for the plugin driver in the guest to access the network adapter hardware directly to send and receive packets, bypassing the host domain and associated overheads. This is illustrated in Figure 1.

The netfront and netback drivers each accept an "accelerator plugin." The frontend accelerator communicates with netfront and implements the data path, whereas the backend accelerator communicates with netback and handles control functions.[2] See Section 3 for details of our implementation of accelerators for the Solarstorm SFC4000 controller.

Accelerated Transmit Path. Packets to be transmitted are passed from the guest kernel to the netfront driver. If an accelerated plugin is present it is given the opportunity to send the packet via the fast path, and it indicates whether or not it did so. If it did not, the netfront driver transmits the packet via the slow path through netback. Thus netfront need have no knowledge of the capabilities of the accelerator. Packets that are destined for local VMs (those in the same system) must not be transmitted onto the network and so in most cases are sent via the slow path. However, some smart adapters might be able to deliver them via an accelerated path.

[2] The backend accelerator can be part of the network adapter's 'net' driver, or can be a separate module.

Accelerated Receive Path. How the receive path is accelerated depends on the capabilities of the hardware and frontend accelerator. Some smart network adapters are able to demultiplex received packets to the appropriate guest by inspecting headers and delivering packets directly into the guest's address space. The frontend accelerator may then be invoked by an interrupt or via the host domain and an event channel. It then passes received packets to the kernel stack. The plugin interface allows the frontend accelerator to participate in Linux's NAPI algorithm for managing interrupts.

If the network adapter is not able to deliver a packet directly to a guest it can deliver it to the net driver in the host domain, from where it will be passed along the normal path through the bridge and netback drivers. This path will usually be used for multicast and broadcast packets that need to be delivered to multiple guests. When a received packet is delivered via netback, it is presented to the backend accelerator. This gives the accelerator an opportunity to inspect the packet and if appropriate program the network adapter to deliver similar packets directly to the guest in future. This provides a means to support hardware with differing means of demultiplexing received traffic and also for the backend accelerator to allocate scarce hardware resources according to demand.

By placing these small hooks in the netfront/netback architecture, we aim to leave sufficient flexibility for device vendors to construct a wide range of acceleration devices and drivers. That is, our goal is to make the netfront/netback drivers in Xen simple and device-agnostic. This keeps the acceleration-specific and device-specific code in the accelerator plugins and thus achieves maximum flexibility.[3]

The code that implements the framework for this architecture was recently accepted by XenSource into the xen-unstable open source repository and will likely be included in future releases of Xen.

2.3 Security

The security concerns that this architecture raises are largely to do with the guest OS's ability to access network hardware directly. It must not be possible for software running in the guest VM, whether by accident or attack, to compromise the security, integrity and isolation guarantees normally provided by Xen. To prevent this, the network adapter must be able to provide the following guarantees:

1. Guests can only transfer network data to and from the adapter using memory regions they own. This may be enforced by the use of a platform IOMMU, or by pre-registering buffers through the host domain (which can do the necessary checks) and restricting access to that set.

[3] Once several accelerator plugin drivers have been developed, it may be that common code is identified and a framework is built to encapsulate this common code. However, that is beyond the current scope of this work.

2. Packets must only be delivered via a fast path if they would have reached the same recipients via the slow path. I.e. packets must only be delivered to their addressee.
3. Guests can only transmit packets which have the correct source MAC address.

Other features may also be desirable, including filtering packets according to other address fields such as VLAN tag or IP address. The ability to control the rate at which individual guests are able to transmit can also be useful to achieve fair access to the network amongst competing VMs.

2.4 Migration

Exposing entire devices directly to the guest (by "PCI pass-through" for example) would render migration between machines with heterogeneous hardware very difficult. Fortunately, the use of the para-virtualized netfront/netback driver model makes migration relatively straightforward.

The plugin model and ability to fall back to the slow path are key. Before a VM is moved from one host to another the frontend accelerator plugin is removed and all network traffic reverts to the slow path. The VM no longer has direct access to the hardware and so can be migrated without difficulty. Note that the removal of the frontend accelerator is largely transparent to applications in the guest VM, which at worst see a degradation in network performance.

On arrival at the new host, the netfront/netback channel is created and networking on the slow path resumes. If the new host contains a smart adapter a suitable frontend accelerator is loaded once again and networking can be accelerated.

The acceleration architecture must also be robust as other VMs migrate out-of and into a host that contains an accelerated VM. When another VM migrates in, the existing frontend accelerators need to be aware of this transition as they must switch from accepting packets on the fast path to sending them down the slow path so they can be delivered locally.[4] Without this it would continue to route packets via the fast path onto the network when the destination is now on the same host.

This is the same problem as that facing Ethernet switches in the network, as they must also become aware of the change in topology. The solution takes advantage of the gratuitous ARP packet sent after migration to update the switches. On receipt of this ARP the backend accelerator plugin can inform the frontend plugin of the change.

3 Implementation for Solarstorm Controllers

Presently the Solarstorm SFC4000 is the only device for which accelerator plugins have been written, but it is likely that other adapters will be supported in the future, particularly with the advent of platform IOMMUs [5,6] and PCI-IOV [7] devices.

[4] Unless the smart adapter itself supports local deliver in hardware.

3.1 Hardware

Like most modern network adapters, the SFC4000 uses DMA descriptor rings to manage transfer of packet data between host memory and the adapter. It has another type of queue, called an *event queue*, to notify the software as DMA transfers complete. In common with some other high-end adapters, the SFC4000 supports many DMA queues and event queues. A transmit ring, receive ring and event queue together form a virtual interface (VI) which comprises all of the resources needed to transfer packets between the host and network. Each VI is accessed through a separate page of the I/O address mapping of the adapter and so an unprivileged domain can be given access to just one VI by mapping the I/O page into the domain's address space.

The SFC4000 steers received packets to the appropriate VI's receive queue by inspecting address fields in packet headers, under the control of *filters*. Filters can only be programmed by the backend accelerator in the priviliged host domain.

In contrast to most other network adapters, the addresses of DMA buffers in host memory can be specified either with physical addresses, or protected virtual addresses. The virtual addressing mode is used for VIs that are mapped into unprivileged domains and provides the memory protection guarantee discussed in Section 2.3. An internal IOMMU is used to translate the virtual addresses to physical addresses, which can only be programmed by the privileged driver.

3.2 Accelerated Transmit

Accelerated transmit is relatively straightforward. Most packets offered by netfront to the accelerator plugin are simply appended to the transmit ring of its VI. However, the frontend plugin elects not to transmit packets addressed to other guests on the same host, including broadcast and multicast packets (netfront will then send them in the normal way). Once the DMA transfer for a send is completed, the adapter places a completion notification on the event queue, which the frontend accelerator responds to by releasing the packet buffer.

3.3 Accelerated Receive

Each packet received by the adapter is directed to a DMA queue according to its addressing information. If the packet's destination address matches a filter, the packet is received onto the VI with which that filter is associated. If the destination address matches no filters, the packet is received onto the default VI.

Packets received onto the default VI queue are handled by the net driver. These packets are passed to the backend accelerator plugin in order to give it an opportunity to accelerate future traffic with that addressing information. The backend accelerator will typically insert a filter to ensure that future packets received to this address are steered by the hardware to the appropriate VI for the guest. The net driver then passes the received packet up to the network stack as usual, which will then make its way over netback to the appropriate netfront in the traditional Xen way.

Currently Xen does not support MSI-X, so all interrupts are delivered to the net driver, even for accelerated traffic. The net driver must then notify the appropriate frontend accelerator (via the backend accelerator) using an event channel.[5] Upon receiving the event channel interrupt, the guest's frontend accelerator inspects its event queue and delivers received packets to the guest OS.

3.4 Performance

We used the Chariot [8] benchmark suite to measure the TCP bandwidth achievable with 1–4 guests, and compared the SFC4000 accelerator plugin implementation against the same (but unmodified) version of the latest open source xen-unstable tree. The experiments were run on two Dell PowerEdge 2950 servers, each fitted with two quad-core Intel Xeon 2.66 GHz CPUs. The CPUs were partitioned such that each VM was pinned to its own (exclusive) physical CPU core and each virtual machine had 256 MB of memory. The servers were connected back-to-back using a pair of Solarstorm SFE4003 CX-4 10 Gb/s Ethernet adapters. The operating systems used were based on Red Hat Fedora Core 5 modified to use the open source xen-unstable tree with Linux 2.6.18-xen kernels. The MTU was 1500 bytes. For tests involving multiple guests, each guest was running the Chariot benchmark client concurrently. All throughput results are the aggregate for all guests.

Table 1 shows results for a single uni-directional TCP stream to each guest. Performance saturates at 9.25 Gb/s due to a PCIe bus bandwidth limitation. Table 2 shows corresponding results for bi-directional traffic.

Table 1. Bandwidth (Gb/s) for uni-directional traffic

Number of guests	Unaccelerated throughput Gb/s	Accelerated throughput Gb/s
1	1.93	5.26
2	2.60	8.86
3	2.80	9.25
4	2.84	9.25

We also noted that under heavy load with unmodified xen-unstable the host domain became very unresponsive to the point of being very difficult to use. This is thought to be due to it becoming overloaded by the network traffic passing through it. In the accelerated experiments there was no human-observable effect on the host domain's responsiveness as the network load was carried by the guests' CPUs.

This illustrates a further benefit of accelerated virtualized networking: Each guest has an independent network stack and so network traffic passed to and from the physical network is multiplexed only in the hardware. Thus there is

[5] We plan to implement MSI-X support in the near future, meaning that the hardware will be able to deliver interrupts directly to the guest.

Table 2. Bandwidth (Gb/s) for bi-directional traffic

Number of guests	Unaccelerated throughput Gb/s	Accelerated throughput Gb/s
1	2.13	5.34
2	2.66	9.95
3	2.85	11.61
4	2.91	14.22

no cross-talk between CPUs that are running separate guests and performance scales very well as the number of CPU cores and guests increases.

4 Solarstorm and User-Level Networking

The same hardware features that make the Solarstorm adapters suitable for direct access by untrusted guest VMs can be used to allow direct access by untrusted user-level applications. We have developed a TCP stack which can be linked against user-level applications and allows the bulk of network processing to be performed in the context of the application, eliminating system calls from the transmit and receive data paths. This results in significant reduction in

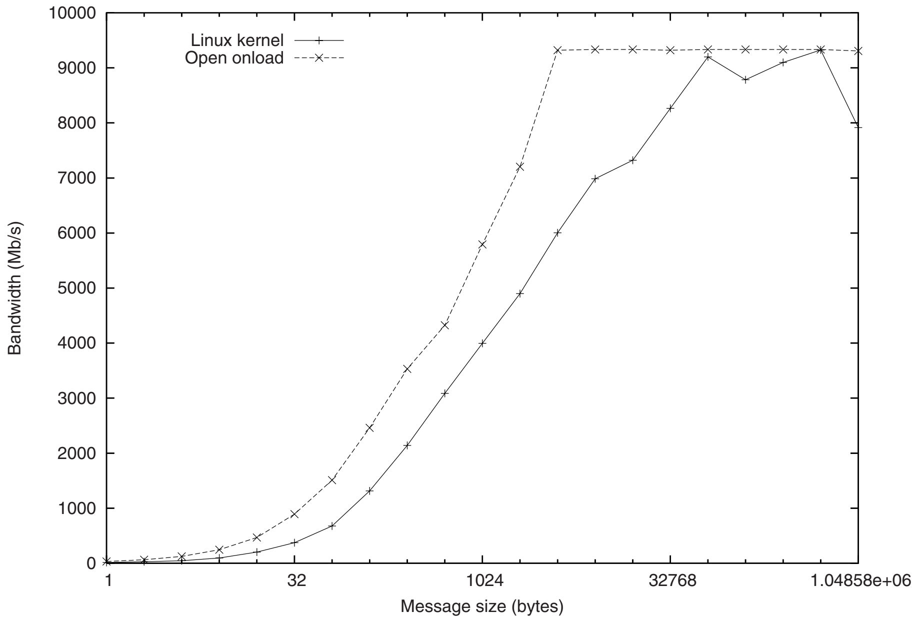


Fig. 2. Bandwidth (Mb/s) of user-level and kernel networking

CPU utilisation, increased bandwidth and reduced latency. We call our user-level TCP/IP stack *Open Onload*, and intend to release it under an open-source licence in the near future.

The plot in Figure 2 shows the bandwidth achieved by Open Onload compared with the Linux kernel stack running over the same hardware. This test was run with a 1500byte MTU and reaches link saturation bandwidth.

The application to application one-way latency for the kernel stack[6] on these systems is 10.1 μs and just 5.3 μs with Open Onload. Of this, 4.3 μs is attributable to the hardware and link, so the software overhead to send and receive a small message is 5.8 μs for the kernel stack and just 1.0 μs for Open Onload.

As well as improving performance, the Open Onload stack improves system behaviour by performing network processing in the context of the application. This ensures that the work done is properly accounted to the application and so improves fairness. Another consequence is that it does not suffer from receive live-lock [9], a situation in which a CPU spends all of its time processing network traffic at high priority and not making any progress in the user-level application. Thus performance under overload conditions is improved.

5 Conclusion

This paper has presented an architecture for accelerating network access from virtual machines. The architecture has several advantages:

Fast. The performance delivered with the acceleration architecture far exceeds that seen by "vanilla" Xen, particularly when many guests are competing for network access. By removing the shared host domain from the common data path, performance isolation is improved.

Simple. The amount of additional code in the Xen mainline has been kept low.

Flexible. The architecture makes minimal assumptions about the capabilities offered by hardware, meaning it should be compatible with a wide range of devices, present and future.

Mobile. The ability to fall-back to the slow path means it is possible to support migration in heterogeneous environments and benefit from acceleration when suitable hardware is available.

Safe. With suitable hardware support the architecture ensures that VMs remain isolated from each other and can not exploit their direct access to the NIC to interfere with other guests.

Using this architecture Xen can now fill a 10 Gb/s pipe and deliver performance to guest OSs comparable to that which is normally only seen by the host OS. Further developments currently in progress, including MSI-X support, are expected to improve performance yet further.

[6] With interrupt moderation disabled.

References

1. Pope, S., Riddoch, D.: 10Gb/s Ethernet performance and retrospective. Computer Communication Review 37(2) (2007)
2. Pratt, I., Fraser, K., Hand, S., Limpach, C., Warfield, A., Magenheimer, D., Nakajima, J., Mallick., A.: Xen 3.0 and the Art of Virtualization. In: Proceedings of the Ottawa Linux Sumposium (2005)
3. Menon, A., Cox, A., Zwaenepoel, W.: Optimizing Network Virtualization in Xen. In: USENIX Annual Technical Conference (2006)
4. Menon, A., Santos, J.R., Turner, Y., Janakiraman, G.J., Zwaenepoel, W.: Diagnosing Performance Overheads in the Xen Virtual Machine Environment. In: ACM/USENIX Conference on Virtual Execution Environments (VEE 2005) (June 2005)
5. Intel Corportation: Intel Virtualization Technology for Directed I/O (2006), `http://www.intel.com/technology/itj/2006/v10i3/2-io/5-platform-hardware-support.htm`
6. AMD Inc: AMD I/O Virtualization Technology (IOMMU) Specification. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf`
7. PCI SIG: PCI-SIG - I/O Virtualization, `http://www.pcisig.com/specifications/iov/`
8. Ixia Corporation: IxChariot., `http://www.ixiacom.com/products/display.php?skey=ixchariot`
9. Mogul, J.C., Ramakrishnan, K.: Eliminating receive livelock in an interrupt-driven kernel. ACM Transactions on Computer Systems 15(3), 217–252 (1997)

Author Index